DISEÑO Y DESARROLLO DE UN SERVICIO BIG DATA EN LA NUBE PARA BÚSQUEDA, COMPARTICIÓN DE FICHEROS Y DATA MINING

Juan Ramos Díaz

GRADO EN INGENIERÍA INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin grado

Junio 2016

Directores: Victoria López, José Javier García Aranda

Autorización de difusión y utilización

Juan Ramos Díaz

Madrid, 17 de Junio de 2016

El abajo firmante, matriculado en el Grado en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "DISEÑO Y DESARROLLO DE UN SERVICIO BIG DATA EN LA NUBE PARA BÚSQUEDA, COMPARTICIÓN DE FICHEROS Y DATA MINING", realizado durante el curso académico 2015-2016 bajo la dirección de Victoria López en el Departamento de Arquitectura de Computadores, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

En esta memoria se presenta el diseño y desarrollo de una aplicación en la nube destinada a la compartición de objetos y servicios.

El desarrollo de esta aplicación surge dentro del proyecto de I+D+i, SITAC: Social Internet of Things – Apps by and for the Crowd ITEA 2 11020, que trata de crear una arquitectura integradora y un "ecosistema" que incluya plataformas, herramientas y metodologías para facilitar la conexión y cooperación de entidades de distinto tipo conectadas a la red bien sean sistemas, máquinas, dispositivos o personas con dispositivos móviles personales como tabletas o teléfonos móviles. El proyecto innovará mediante la utilización de un modelo inspirado en las redes sociales para facilitar y unificar las interacciones tanto entre personas como entre personas y dispositivos.

En este contexto surge la necesidad de desarrollar una aplicación destinada a la compartición de recursos en la nube que pueden ser tanto lógicos como físicos, y que esté orientada al big data.

Ésta será la aplicación presentada en este trabajo, el "Resource Sharing Center", que ofrece un servicio web para el intercambio y compartición de contenido, y un motor de recomendaciones basado en las preferencias de los usuarios.

Con este objetivo, se han usado tecnologías de despliegue en la nube, como Elastic Beanstalk (el PaaS de Amazon Web Services), S3 (el sistema de almacenamiento de Amazon Web Services), SimpleDB (base de datos NoSQL) y HTML5 con JavaScript y Twitter Bootstrap para el desarrollo del front-end, siendo Python y Node.js las tecnologías usadas en el back end, y habiendo contribuido a la mejora de herramientas de clustering sobre big data. Por último, y de cara a realizar el estudio sobre las pruebas de carga de la aplicación se ha usado la herramienta ApacheJMeter.

Palabras clave

Nube, Internet de las Cosas, Big Data, Compartición, Clustering, Python, Node.js, Elastic Beanstalk, Amazon Web Services, SimpleDB.

Abstract

This paper presents the design and development of a cloud application intended for sharing objects and services.

The development of this application comes within the I+D+i, SITAC: Social Internet of Thing – Apps by and for the Crowd ITEA 2 11020, trying to create an integrated architecture and an "ecosystem" that includes platforms, tools and methodologies to facilitate the connection and cooperation of entities of different types connected to the network, whether systems, machines, devices or people with personal mobile devices such as tablets or mobile phones. The project will innovate by using a model inspired by social networks to facilitate and unify the interactions both among people and between people and devices.

In this context arises the need to develop an application for the sharing of resources in the cloud that can be both logical and physical, and that is oriented big data.

This will be the application presented in this paper, the "Resource Sharing Center" which offers a web service for the exchange and sharing of content, and a recommendation engine based on user preferences.

To this end, it has been used cloud deployment technologies, as Elastic Beanstalk (PaaS service form Amazon Web Services), S3 (storage service form Amazon Web Service), SimpleDB (NoSQL database), and HTML5 with JavaScript and Twitter Bootstrap for the front-end development, with Python and Node.js being used for the back-end development, and having contributed to improving clustering tools on big data. Finally, and in undertaking the study load testing the application it has been used ApacheJMeter tool.

Keywords

Cloud, Internet of Things, Big Data, Sharing, Clustering, Python, Node.js, Elastic Beanstalk, Amazon Web Services, SimpleDB.

Índice

Índice de Figuras

1 Introducción

1.1 Objetivos

El objetivo de este trabajo es la realización de una aplicación que dentro del contexto del internet de las cosas (IoT) con los paradigmas de compartición de recursos en redes sociales, basados en los "círculos" o "comunidades" donde un recurso es compartido, permitiendo a los usuarios de la plataforma compartir y gestionar recursos físicos (sensores y actuadores, plazas de aparcamiento, servicios de gestión del aire acondicionado, etc.) y lógicos (documentos multimedia electrónicos, servicios web, enlaces, etc.).

Es bien conocido el desarrollo actual del internet de las cosas debido al desarrollo de las nuevas tecnologías, especialmente las tecnologías big data [1] [2].

De cara a esto se ha desarrollado una aplicación web destinada al alojamiento de recursos, en la que prima la compartición de estos ya sea de forma gratuita o de pago. La aplicación ofrece estos recursos a los usuarios permitiéndoles la posibilidad de buscar ellos mismos los recursos, y también ofreciéndoles recomendaciones basadas en sus gustos, para esto último es necesario crear perfiles de los usuarios basados en sus preferencias y los recursos que comparten, y a partir de esto generar un motor de recomendaciones basado en clustering que ofrezca recursos que puedan interesar al usuario.

Esta aplicación es accesible desde cualquier dispositivo, por lo que ha primado el diseño web adaptable (*responsive web design*) para que la aplicación sea igual de accesible desde un móvil hasta un PC.

También se ha realizado un estudio y análisis de la elasticidad del servicio, destinado a tener un uso masivo, para esto también se adaptara el motor de recomendación, que se ha diseñado teniendo en mente un gran volumen de usuarios.

1.2 Estructura de la memoria

Esta memoria se compone de 7 capítulos siendo los capítulos centrales que describen el desarrollo y funcionalidad del proyecto los capítulos 3 y 4.

El capítulo 3 describe y detalla el funcionamiento del Resource Sharing Center, que es el servicio en el que se basa este proyecto. En este capítulo, lo primero que se explica es la arquitectura del servicio y los principios de diseño que se han seguido, luego se introducen los componentes del servicio que más adelante serán descritos en detalle y por último se define el modelo de datos utilizado.

El capítulo 4 describe y detalla el desarrollo del módulo Analytics, en el que se realiza un proceso de clustering sobre los usuarios del Resource Sharing Center para desarrollar un motor de recomendaciones. En este capítulo, primero se explicará la definición de los clusters y la

función distancia entre ellos, después la selección del algoritmo y la librería usada que lo implemente, y por último se define el modelo de datos usado.

El resto de capítulos son, el estado del arte (capítulo 2), los resultados de este trabajo (capítulo 5), las conclusiones obtenidas y líneas futuras de trabajo (capítulo 6), y por último un anexo en el que se detallan las herramientas usadas durante el desarrollo de este trabajo, y se muestran en detalle los aspectos más importantes de la implementación.

2 Estado del arte

2.1 Computación en la nube

La computación en la nube, también conocida como servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos, es un modelo que permite el acceso ubicuo, conveniente y en demanda a una red compartida de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser provistos o entregados con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios.

Basándose en este modelo se distingue entre las plataformas de computación en la nube, y los servicios en la nube, siendo estos primeros una colección de servicios e infraestructura que permiten a sus consumidores gestionar y garantizar los servicios en la nube [3].

2.1.1 Características esenciales

- Autoservicio en demanda: El consumidor se puede proveer de forma automática de recursos de computación como almacenamiento en la red, según sea necesario sin interacción humana con el proveedor de servicio.

- Amplio acceso a la red: Los servicios están disponibles sobre la red y se accede mediante mecanismos que promueven el uso de plataformas heterogéneas (teléfonos móviles, tabletas, ordenadores de sobremesa o portátiles).

- Puesta en común de recursos: Los recursos del proveedor, tanto lógicos como físicos, son puestos en común y asignados o reasignados a un gran número de consumidores en función de la demanda, y sin que ellos tengan que intervenir de forma explícita. Estos servicios suelen ser de almacenamiento, procesamiento, memoria o ancho de banda.

- Elasticidad: Los recursos informáticos son provistos de forma elástica, es decir aprovisionados y liberados en función de la demanda, de forma que para el consumidor esta provisión de recursos parezca ilimitada y se pueda asignar en cualquier momento.

- Servicio medido y personalizado: Los sistemas en la nube pueden controlar y optimizar el uso de recursos automáticamente. Este uso de servicios puede ser monitorizado, controlado y entregado al consumidor estableciendo una relación de transparencia para ambos.

2.1.2 Modelos de servicio

Las plataformas de computación en la nube pueden ofrecer sus servicios de las siguientes maneras.

- Software como servicio (SaaS): El consumidor de la nube no gestiona ni controla la propia infraestructura (red, servidor, sistema operativo, almacenamiento). Desarrolla el servicio al margen del despliegue del mismo. Una aplicación ofrecida en modo SaaS se ejecutará sobre una infraestructura PaaS o IaaS

- Plataforma como servicio (PaaS): El consumidor de la nube puede gestionar y controlar el lenguaje de programación, librerías, servicios y otras herramientas que le provee el gestor de la nube para desarrollar su servicio. Pero no tiene control de la capa inferior de la infraestructura de la nube, red, servidor, almacenamiento.

- Infraestructura como servicio (IaaS): El consumidor de la nube puede gestionar y controlar la infraestructura que le ofrece el proveedor de la nube. Puede controlar el sistema operativo, de almacenamiento, el lenguaje de programación y librerías que usara para desarrollar su servicio.

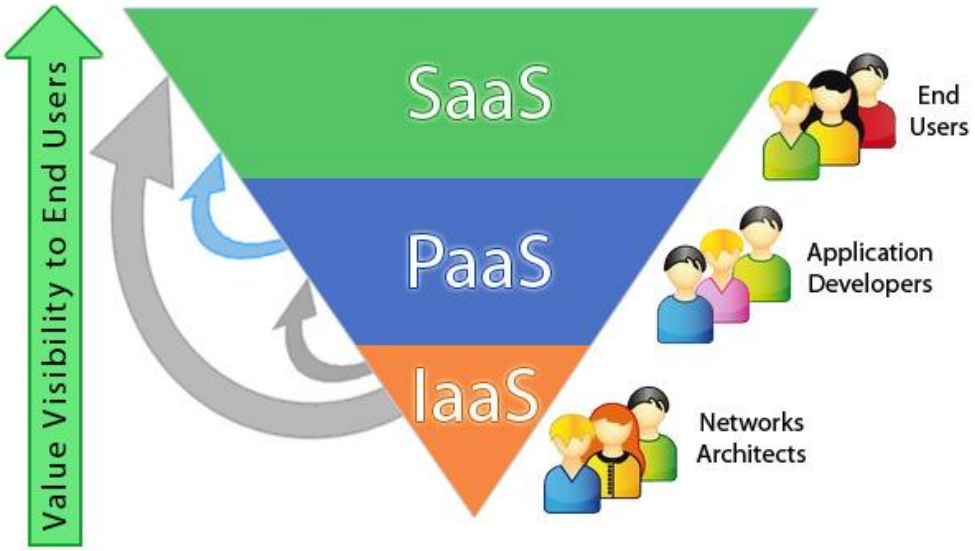

Figura 1. Modelos de servicio

2.1.3 Modelos de despliegue

Existen cuatro modelos de despliegue que se describen a continuación.

- Nube privada: La infraestructura de la nube es suministrada para el uso exclusivo de una única organización que comprende a múltiples consumidores. El dueño de la nube, o encargado de gestionarla puede ser la organización, o una organización ajena, o una combinación entre ambas.

- Nube comunitaria: La infraestructura de la nube es suministrada para el uso exclusivo de una comunidad de consumidores, que pueden ser organizaciones con políticas exclusivas. El dueño de esta nube, o el encargado de gestionarla puede ser una o más de las organizaciones de la comunidad, una organización ajena, o una combinación entre ambas.

- Nube pública: La infraestructura de la nube es suministrada para el uso del público general. El dueño de la nube o el encargado de gestionarla puede ser un centro académico, organización gubernamental, o una combinación de ambas.

- Nube híbrida: La infraestructura de la nube es una composición de dos o más infraestructuras de nube que siguen modelos de despliegue distintos, que pueden seguir siendo entidades separadas, pero establecen políticas para permitir la portabilidad de las distintas infraestructuras.

2.1.4 Plataformas de computación en la nube disponibles

Entre las más importantes se encuentran las siguientes:

- Amazon Web Services (AWS) [4]: La plataforma de computación en la nube de Amazon, es una de las pioneras en este campo y es usada en aplicaciones como Reddit o Netflix. Algunos de sus principales servicios son: Amazon EC2, Amazon S3, y Amazon RDS. Estos servicios están dedicados respectivamente al hosting de aplicaciones, almacenamiento masivo, y gestión de bases de datos.

- Google Cloud Platform [5]: La plataforma de computación en la nube de Google, ofrece a sus clientes la misma infraestructura de soporte que Google usa internamente para sus productos como Google Search y Youtube. Ofrece servicios como Google App Engine, para el hosting de aplicaciones, Google Cloud Storage para almacenamiento, y Google Bigtable para la gestión de bases de datos.

- Microsoft Azure [6]: La plataforma de computación en la nube de Microsoft, en competencia directa con las plataformas de Google y Amazon, es usada por empresas como Mazda o Heineken, ofrece servicios como AzureVM, AzureBackup, y DocumentDB para hosting, almacenamiento y gestión de bases de datos.

- Openshift [7]: La plataforma de computación en la nube de RedHat es código abierto y se puede acceder para desarrollar aplicaciones en GitHub. Ofrece servicios basados en el modelo Platform as a Service (PaaS), para esto da soporte a los entornos de programación más comunes, como Node.js, Ruby o Python, y frameworks de aplicaciones web como WSGI para Python o Rack para Ruby. También soporta programas binarios que sean aplicaciones web, siempre que se puedan ejecutar en la distribución de Linux de RedHat (RHEL).

2.2 Herramientas de Amazon Web Service.

Amazon Web Service (AWS) es la infraestructura en la nube de Amazon, ofrece las herramientas necesarias para desarrollar servicios en la nube para cualquier modelo (PaaS, IaaS, o SaaS). A continuación vamos a enumerar y detallar las principales herramientas que ofrece.

- **Amazon SimpleDB:** Es una base de datos NOSQL (Not Only SQL), que además de garantizar almacenamiento masivo con accesos muy rápidos sin tener que usar operaciones complejas que se dan en bases de datos SQL, permite también al cliente abstraerse de su gestión, administración y mantenimiento de la misma, ya que Amazon administra automáticamente el aprovisionamiento de la infraestructura, mantenimiento de hardware y software, e indexa automáticamente los datos (índice automático de SimpleDB), y crea y administra réplicas de los mismos que distribuye geográficamente para permitir alta disponibilidad y capacidad de duración.
  SimpleDB se adapta también con facilidad a cualquier cambio, se puede añadir cualquier atributo a cualquier tabla sin preocuparse por romper un esquema rígido o refactorizar el código.
  El servicio sólo cobra los recursos consumidos en almacenamiento de los datos y en distribución de las solicitudes, y no cobra nada en la transferencia de datos entre SimpleDB y otros servicios de AWS en la misma región.
  Otros servicios de bases de datos que ofrece AWS son: **Amazon RDS**, base de datos relacional en la nube, que también abstrae al programador de las tareas de administración; **Amazon DynamoDB**, base de datos NOSQL mas escalable que SimpleDB pero menos flexible.

- **Amazon Simple Storage Service (Amazon S3):** Es un servicio de almacenamiento en la nube orientado a ofrecer una alta durabilidad, almacenando los datos de forma sincrónica en varias instalaciones, disponibilidad, permitiendo además la posibilidad de elegir en que región estarán los datos para optimizar la latencia, seguridad, permitiendo transmitir sus datos a través de SSL y cifrarlos de forma automática y mediante su compatibilidad con el servicio de acceso e identidad de Amazon (AWS IAM), escalabilidad, ya que se pueden almacenar todos los datos que se desee sin preocuparse sobre cuanto se va a necesitar, con facilidad de uso, estando integrado en otros servicios de AWS (como Elastic Beanstalk), y ofreciendo API REST completas y los SDK para permitir una integración sencilla con otras tecnologías.
  Amazon S3 busca también reducir el precio cobrando solo por lo que se necesite, sin compromisos de gasto mínimo ni cuotas iniciales, y permitiendo la transferencia de datos que no se usen a menudo a Amazon Glacier reduciendo de esta forma también los costes.

- **Amazon Elastic Compute Cloud (Amazon EC2):** Servicio de hospedaje de servidores virtuales, proporciona recursos computacionales de forma elástica orientada a los servicios de la nube.
  Amazon EC2 permite elegir los recursos sobre los que desplegar servicios en la nube, configuración de memoria, CPU, almacenamiento de la instancia y tamaño de partición de arranque para su sistema operativo y su aplicación.
  Amazon EC2 otorga un control completo sobre los recursos informáticos que se desplieguen en la nube y permite instanciar todas las máquinas necesarias en demanda.
  Para adaptar el precio a las necesidades de cada usuario Amazon ofrece tres modelos de compra de instancias de AmazonEC2: Bajo demanda, reservadas e instancias de subasta. También ofrece la opción de pagar por hosts dedicados.

- **Amazon Route 53:** Servicio web DNS escalable y de alta disponibilidad, conecta las solicitudes del usuario con la infraestructura en ejecución en AWS, es compatible con los servicios de AWS, como Amazon EC2 y Amazon S3, y también puede direccionar a usuarios a infraestructuras externas a AWS.
  Contiene la herramienta Amazon Route 53 Traffic Flow, se puede dirigir el tráfico en función del estado de los puntos de enlace, la ubicación geográfica y la latencia.

- **AWS Elastic Beanstalk:** Permite implementar servicios en la nube siguiendo el modelo PaaS, está diseñado para que sea muy sencillo implementar y escalar servicios y aplicaciones web desarrolladas con distintas tecnologías, preocupándose el usuario de AWS solo de cargar el código ya que Elastic Beanstalk administra de forma automática el servicio, encargándose del aprovisionamiento de la capacidad, equilibrio de carga, escalado automático y monitorización de la aplicación.

- **Amazon Elastic MapReduce (Amazon EMR):** Servicio que facilita el proceso de big data valiéndose de la tecnología Hadoop, orientado al procesamiento de grandes cantidades de datos entre instancias de Amazon EC2 dinámicamente escalables de forma muy sencilla y rentable.
  Amazon EMR también administra el resto de casos de uso de big data, como el análisis de logs, indexación web, almacenamiento de datos, aprendizaje automático, análisis financiero, simulación científica y la bioinformática.

- **AWS Identity and Access Management (Amazon IAM):** Servicio gratuito que permite controlar el acceso de los usuarios a servicios y recursos de AWS. Permite crear usuarios y grupos de usuarios, o roles que estos usuarios puedan asumir. Permite gestionar las políticas de acceso y de confianza en compañías de terceros, tanto para usuarios y grupos como para roles.

2.3 Desarrollo en Elastic Beanstalk

Elastic Beanstalk es el servicio PaaS de Amazon, orientado a facilitar la administración de aplicaciones cloud ya que el desarrollador solo tiene que cargar la aplicación y Elastic Beanstalk administrará de manera automática los detalles de la administración y estado de la aplicación.

Elastic Beanstalk provee plataformas para lenguajes de programación (Java, PHP, Python, Ruby, Go) y contenedores Web (Tomcat, Passenger, Puma). Y los recursos necesarios para ejecutar las aplicaciones, incluyendo instancias de Amazon EC2.

Para usar este servicio, lo primero que se hace es cargar la aplicación en Elastic Beanstalk informando al servicio del lenguaje en que está desarrollada y sobre qué sistema operativo quieres que corra la instancia en Amazon EC2 que contendrá la aplicación. Elastic Beanstalk automáticamente lanza un entorno y crea y configura los recursos de AWS necesarios para ejecutar el código. Una vez que el entorno está lanzado, se puede administrar, desarrollar nuevas versiones de tu aplicación, y de forma opcional también puedes administrar el escalado de las instancias de EC2, el estado del entorno, o manejar eventos y métricas desde la interfaz web de Elastic Beanstalk, como pueden ser los disparadores de elasticidad, para controlar el escalado dinámico del servicio. La Figura 2 muestra el flujo de trabajo de Elastic Beanstalk.

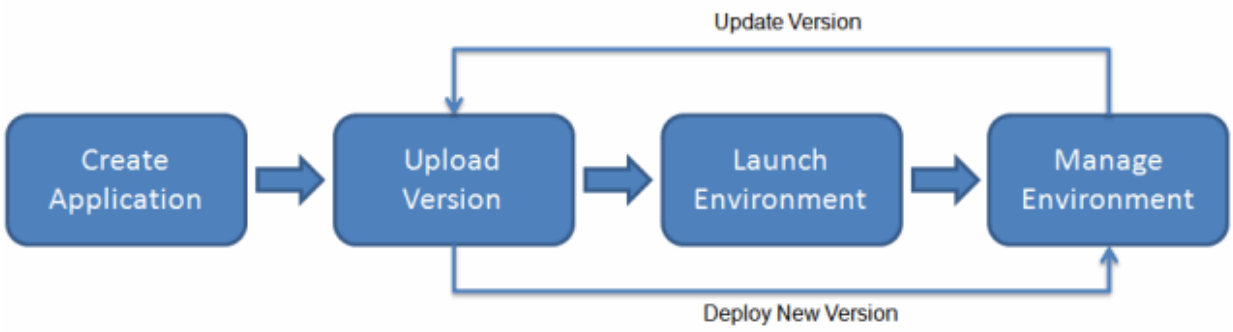

Figura 2: Flujo de trabajo en Elastic Beanstalk

2.3.1 Componentes de Elastic Beanstalk

- **Aplicación:** Una aplicación en Elastic Beanstalk es la colección lógica de los componentes de la misma, incluyendo los entornos, las versiones y configuraciones del entorno. Conceptualmente es similar a una carpeta.

- **Versión de una aplicación:** Una versión de aplicación se refiere al código de una iteración específica de la aplicación web. Cada versión es almacenada como un objeto en Amazon S3 conteniendo el código de la misma.
  En un entorno activo se puede desplegar en cualquier momento cualquier versión de una aplicación ya cargada en S3.

- **Entorno:** El entorno de una aplicación mantiene cargada la versión actual del código y provee a la aplicación de los recursos necesarios para ejecutarla, como las instancias en EC2, que son las máquinas que cargarán el código, monitoriza el servidor de la aplicación, o los balanceadores de carga. La Figura 3 muestra una aplicación con dos entornos.

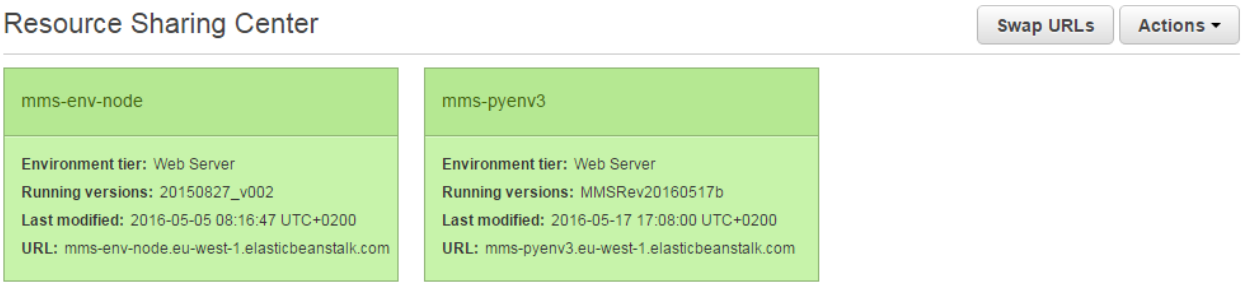

Figura 3: Aplicación de AWS EB con dos entornos

2.3.2 Roles, instancias de perfiles y políticas de usuarios.

Cuando se crea un entorno en Elastic Beanstalk, se debe proporcionar un rol y una instancia de un perfil del servicio de Amazon IAM. El rol permite a Elastic Beanstalk asumirlo para poder usar otros servicios de AWS asociados y la instancia del perfil permite a las instancias dentro del entorno subir informes a S3 y realizar distintas tareas de administración que dependen de cada tipo de entorno y plataforma.

Además se pueden crear políticas de usuarios y aplicarlas en la cuenta de IAM, para crear y administrar entornos de Elastic Beanstalk, esto da mayor control para los distintos escenarios en los que se desarrolla la aplicación, permitiendo al usuario más libertad en la aplicación o añadir más restricciones en distintos casos de uso.

- **Rol de Elastic Beanstalk:** Es el rol que Elastic Beanstalk asume cuando interactúa con otro servicio, como pueden ser permisos adicionales para interactuar con las instancias en EC2 en las que se ejecuta la aplicación. En la Figura 4 vemos un ejemplo de cómo es un rol en Elastic Beanstalk.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:DescribeInstanceHealth",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:GetConsoleOutput",
        "ec2:AssociateAddress",
        "ec2:DescribeAddresses",
        "ec2:DescribeSecurityGroups",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:DescribeNotificationConfigurations"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Figura 4: Ejemplo de Rol en Elastic Beanstalk

- **Instancia de un perfil:** Una instancia de un perfil de Amazon IAM en Elastic Beanstalk es un rol que se aplica a las instancias lanzadas dentro del entorno. Para estas instancias se deben definir los permisos, como pueden ser acceso a Amazon S3, para leer, escribir o ambos. En la Figura 5 vemos un ejemplo de una instancia de un perfil.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "BucketAccess",
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::elasticbeanstalk-*",
        "arn:aws:s3:::elasticbeanstalk-*/*"
      ]
    }
  ]
}
```

Figura 5: Instancia de perfil

- **Política de usuario:** Son usuarios de Amazon IAM creados para cada persona que use Elastic Beanstalk y así evitar que usen una cuenta con permisos de administrador, o que éste comparta sus credenciales.
  Estas condiciones permiten a otro desarrollador con menos permisos desplegar ciertos servicios en Elastic Beanstalk, o encargarse sólo de ciertas tareas de administración. La Figura 6 muestra un ejemplo de estas políticas.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticbeanstalk:*",
                "ec2:*",
                "ecs:*",
                "elasticloadbalancing:*",
                "autoscaling:*",
                "cloudwatch:*",
                "dynamodb:*",
                "s3:*",
                "sns:*",
                "cloudformation:*",
                "rds:*",
                "sqs:*",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:PassRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfiles",
                "iam:ListRoles",
                "iam:ListServerCertificates",
                "acm:DescribeCertificate",
                "acm:ListCertificates"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/aws-elasticbeanstalk*",
                "arn:aws:iam::*:instance-profile/aws-elasticbeanstalk*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iam:PolicyArn": [
                        "arn:aws:iam::aws:policy/AWSElasticBeanstalk*",
                        "arn:aws:iam::aws:policy/service-role/AWSElasticBeanstalk*"
                    ]
                }
            }
        }
    ]
}
```

Figura 6: Ejemplo de Politica de usuario

2.4 Tecnologías de desarrollo en la nube.

Las tecnologías de programación que se han usado en este trabajo y se van a describir en este apartado son: Python y Javascript, y Node.js, otras tecnologías usadas son Ajax, Json y Twitter Bootstrap que también se describen en esta sección.

2.4.1 Python

Python es un lenguaje de programación interpretado de alto nivel, interactivo, y multi-paradigma. Usa tipado dinámico, es multiplataforma, y basa su sintaxis en la indexación, para facilitar la legibilidad del código [8].

Python fue creado en 1991 por Guido van Rossum en el Stichting Mathematisch Centrum en los Países Bajos como sucesor de un lenguaje llamado ABC. Y en 2001 se fundó la Python Software Foundation, una organización sin ánimo de lucro creada específicamente para ser la poseedora de toda la propiedad intelectual relativa a Python. Python se distribuye bajo una licencia de código abierto denominada Python Software Foundation License, compatible con la licencia general GNU [9].

2.4.1.1 Principales características.

- Multiparadigma: Python no fuerza al desarrollador a programar de una forma determinada si no que admite distintos paradigmas. Admite completamente los paradigmas de programación orientada a objetos y de programación estructurada. Y en menor medida también soporta muchas características de otros paradigmas como programación funcional (lista intensionales y diccionarios), programación orientada a aspectos y mediante extensiones se pueden incorporar más paradigmas.

- Zen of Python [10]: The zen of Python es un documento que resume y da indicaciones sobre la filosofía de programación Python, en la que prima un código claramente legible, y fácil de entender.

- Facilidad de extensión: Python está diseñado para ser fácilmente extensible antes que construir toda la funcionalidad en el núcleo del lenguaje. Es muy fácil construir y usar extensiones en C y C++ que cubran o mejoren funcionalidades de Python, como puede ser la optimización de la velocidad, permitiendo así mantener la claridad del lenguaje en lugar de complicarlo.

- Gestión de la memoria: Python optimiza la gestión de la memoria usando tipado dinámico y conteo de referencias para liberar los recursos. Además usa resolución dinámica de nombres, enlazando nombres de variables y métodos durante la ejecución del programa.

2.4.1.2 Desarrollos Python en AWS

El servicio de Amazon Elastic Beanstalk ofrece soporte para el lenguaje Python en todas sus versiones, proporcionando la librería Boto para interactuar con los servicios de AWS, y por su parte Python ofrece diversos frameworks con los que implementar su parte servidora.

- Boto: Es la API (interfaz de programación de aplicaciones, que ofrece una capa de abstracción para usar una librería por otro software) que ofrece AWS a los usuarios de Python actualmente.
  Boto dispone de dos niveles distintos de API. Las API de cliente, que proporcionan asignaciones especificas a las operaciones HTTP. Y las API de recursos que ocultan llamadas a la red específicas, pero permiten obtener recopilaciones y objetos de recursos para acceder a atributos y realizar acciones [11]. En la Figura 7 vemos un fragmento de código en Python en el Boto permite que se interactúe con EC2.

```
for i in ec2.instances.all():
    if i.state['Name'] == 'stopped':
        i.start()
```

Figura 7: Ejemplo de uso de la librería Boto en Python

- Flask: Es un microframework (marco de trabajo minimalista orientado a aplicaciones web), que a pesar de no contener algunas aplicaciones que si contienen frameworks completos, como puede ser una capa de compatibilidad con bases de datos, contiene la funcionalidad necesaria para servicios web y un gran soporte para extensiones que apliquen las funcionalidades de que carece [12]. En la Figura 8 vemos un fragmento de código que muestra la interacción entre Flask y Boto.

```
from flask import Flask
from flask import request
from flask import jsonify
import json

import boto
from boto.sts import STSConnection
import boto.sdb

def getAWSToken(sitac_token):
        sts_connection = boto.sts.connect_to_region('eu-west-1',aws_access
```

Figura 8: Interacción entre Flask y Boto

2.4.2 Javascript

Es un lenguaje de programación de alto nivel, interpretado, dinámico, débilmente tipado, y multiparadigma. Ha sido estandarizado en la especificación del lenguaje ECMAScript. Junto con

HTML y CSS es una de las tres tecnologías que componen el núcleo del desarrollo en la web [13].

Javascript fue desarrollado originalmente en Mayo de 1995 por Brendan Eich, con la intención de ser un lenguaje interpretado orientado a programadores no profesionales [14]. En Septiembre de 1995 fue integrado en el navegador NetScape 2.0B3 [15]. En 1997 los autores propusieron que JavaScript fuera adoptado como estándar de la European Computer Manufacturers 'Association ECMA, esto sucedió en Junio de 1997 con el nombre de ECMAScript. Actualmente se cerró y publico el estándar ECMAScript 6 en Junio de 2015 [16].

2.4.2.1 Características principales

- Imperativo y estructurado: JavaScript es similar a C en cuanto a la estructura de programación, pero difiere de este en cuanto al ámbito de las variables, ya que en Java se mantiene en la función, hasta que en la versión de Java Script 1.7 se añade el ámbito por bloques con la palabra reservada "let" [17].
- Dinámico: En Java Script el tipo de una expresión está asociado al valor y no a la variable, por lo que una misma variable puede tener varios tipos distintos a lo largo de la ejecución [18].
- Orientación a objetos por prototipos: Java Script implementa la herencia de clases mediante prototipos, que también permiten emular el comportamiento de la orientación a objetos mediante clases [19].
- Lado servidor: JavaScript está destinado a ejecutarse en el lado cliente de las aplicaciones, pero hay implementaciones en el lado servidor, siendo Node.js la más destacable.

2.4.2.2 Desarrollo Javascript en AWS

Amazon AWS ofrece un kit de software para desarrolladores (AWS SDK for JavaScript) que permite implementar aplicaciones o librerías que usan servicios de AWS usando una sencilla API, valida tanto para desarrollos en el front-end, o en aplicaciones en el servidor basadas en Node.js [20].

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-2.3.14.min.js"></script>
```

Figura 9: Inserción del SDK de AWS en una página

2.4.3 Node.js

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript (JavaScript), asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor de JavaScript V8 de Google [21].

Node.js fue desarrollado originalmente por Ryan Dahl en 2009 y fue patrocinado por Joyent [22]. Al principio sólo estaba soportado por Linux hasta que en Junio de 2011 Microsoft implementó una versión nativa de Windows [23]. También en 2011 se introdujo el gestor de paquetes *npm*, permitiendo a más programadores y desarrolladores publicar y compartir fuentes de librerías de Node.js, y otras actualizaciones [24].

El desarrollo de Node.js se dividió entre Node.js y Ios.js en Diciembre de 2014 debido a conflictos internos. En Junio de 2015 se fundó la Node.js Foundation [25] para que los desarrollos de Node.js e ios.js trabajasen conjuntamente hasta que en septiembre de 2015 se volviesen a juntar de nuevo [26].

Las principales características son:

- Paralelismo: Node.js trabaja con un único hilo y usando llamadas de entrada-salida no bloqueantes, permitiendo así la posibilidad de realizar un gran número de llamadas concurrentes, sin incurrir en costes de gasto de contexto, y usando siempre una función de *callback*, para continuar con la ejecución y seguir realizando llamadas. [27]
- Motor V8: V8 es el entorno de ejecución para JavaScript que usa Google Chrome, y sobre el que se ejecuta Node.js, cuyo núcleo de operaciones está escrito en JavaScript, con métodos de soporte en C++.
- Lazo de eventos: En Node.js cada conexión recibe espacios de memoria en lugar de generar un hilo de trabajo, y cada vez que se realiza una conexión se ejecuta un callback. El lazo de eventos no es llamado explícitamente, en su lugar, se activa después de cada ejecución de callback y finaliza cuando no quedan eventos que atender [28].

2.4.3.1 Desarrollo AWS con Node.js

El servicio de Amazon Elastic Beanstalk ofrece soporte para Node.js, proporcionando un kit de software para desarrolladores (AWS SDK para JavaScript en Node.js) para interactuar con los servicios de AWS, y también permite el uso de frameworks de Node.js para personalizar la aplicación.

- Express: Express es una infraestructura de aplicaciones web Node.js minimalista y flexible que proporciona una delgada capa extra sobre Node.js para facilitar el desarrollo y permitir seguir usando la funcionalidad de Node.js [29]. Contiene una extensión llamada Jade [30] que permite extiende la funcionalidad de Express permitiendo el uso de plantillas.

- AWS SDK: El kit de software para desarrolladores que ofrece AWS, que reduce la complejidad del desarrollo al proporcionar objetos JavaScript para los servicios en AWS [31]. El uso es como el del SDK de JavaScript, pero indicándolo también en el fichero package.json de la aplicación para así mantener la coherencia estructural, como se puede ver en la figura 10.

Figura 10: Ejemplo de package.json

2.4.4 Otras tecnologías

En esta sección se describen el resto de tecnologías de desarrollo de aplicaciones en la nube usadas en este trabajo.

AJAX

Ajax (Asynchronous JavaScript And XML), es una técnica de desarrollo web que permite desarrollar aplicaciones en el navegador que se comunican de forma asíncrona con el servidor de forma transparente para el usuario. Permite que la página se recargue de forma automática y aumenta la interactividad y usabilidad de las aplicaciones web [32].

El termino AJAX fue acuñado por primera vez en 2005 por James Garret [33], pero el desarrollo de estas técnicas de scripting remoto se remonta a una década antes con Microsoft que introdujo el elemento "iframe" para permitir cargas asíncronas, y en 1998 con Microsoft Remote Scripting. Antes de que el objeto XMLHttpRequest se asentase en la mayoría de navegadores se seguía usando Microsoft Remote Scripting y JavaScriptOnDemand [34].

Tecnologías que utiliza:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño de la parte cliente.
- Document Object Model (DOM) al que se accede mediante JavaScript para interactuar con la información mostrada.
- El objeto XMLHttpRequest, que se usas para el intercambio de datos asíncrono con el servidor web.
- XML, JSON, son los formatos más usados para la transferencia de datos desde el servidor.

JSON

JSON (JavaScript Object Notation), es un formato de texto ligero usado para el intercambio de datos que se ha impuesto frente a otros formatos por la sencillez para ser leído por humanos, por la facilidad con la que se puede programar un parser (analizador sintáctico) y, debido a la ubicuidad de JavaScript en casi cualquier navegador web, es el formato de intercambio de datos más usados en intercambios mediante AJAX. En realidad es un subconjunto de la notación literal de objetos de JavaScript, pero debido a su amplia adopción se considera un formato de texto independiente.

JSON fue especificado por Douglas Crockford a principios de la década del 2000 [35], debido a la necesidad de implementar una comunicación servidor-navegador sin depender de extensiones. Junto con JavaScript, Http básico y Html permitía desarrollar aplicaciones web en tiempo real de forma muy sencilla, lo que le hizo ganar popularidad, y que a pesar de ser un subconjunto de JavaScript, en 2013 se oficializó en el estándar ECMA-404 [36]. En la Figura 11 vemos un ejemplo de un objeto en formato JSON.


```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                {
                    "value": "New", "onclick": "CreateNewDoc()"
                },{
                    "value": "Open", "onclick": "OpenDoc()"
                },{
                    "value": "Close", "onclick": "CloseDoc()"
                }
            ]
        }
    }
}
```


Figura 11: Ejemplo de objeto JSON

Twitter Bootstrap

Twitter Bootstrap es un front-end web framework, es un conjunto de herramientas de diseño para aplicaciones y sitios web. Está orientado al diseño web adaptable, permitiendo programar las mismas plantillas para ser vistas desde distintos dispositivos y está basado en HTML y CSS con extensiones adicionales en JavaScript

Fue desarrollado por Mark Otto y Jacob Thornton de Twitter, con la idea de crear un framework que diese consistencia visual a los distintos desarrollos internos. En 2011 como parte de una hackaton interna de Twitter se liberó el código fuente del proyecto como un proyecto de código abierto [37], y se encuentra actualmente en el desarrollo de su cuarta versión [38]. En la Figura 12 se puede ver un ejemplo de página desarrollada con Twitter Bootstrap.

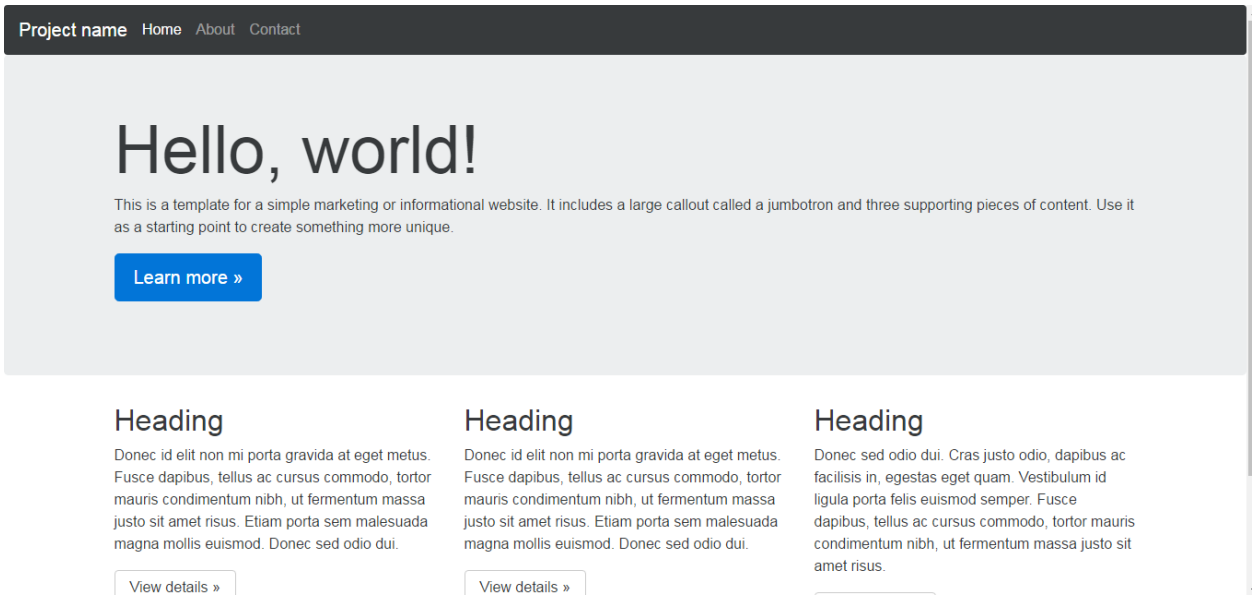

Figura 12: Página desarrollada con Bootstrap

Apache JMeter

Apache JMeter es una aplicación de código abierto diseñada para realizar pruebas de carga, y realizar medidas de rendimiento, originalmente creado para ser probada en aplicaciones web, se ha expandido para realizar medidas de cualquier otro tipo de funciones, como son recursos web estáticos y dinámicos, servidores FTP y también puede simular cargas muy pesadas en servidores y redes para determinar su rendimiento y analizarlo frente a diferentes cargas [39].

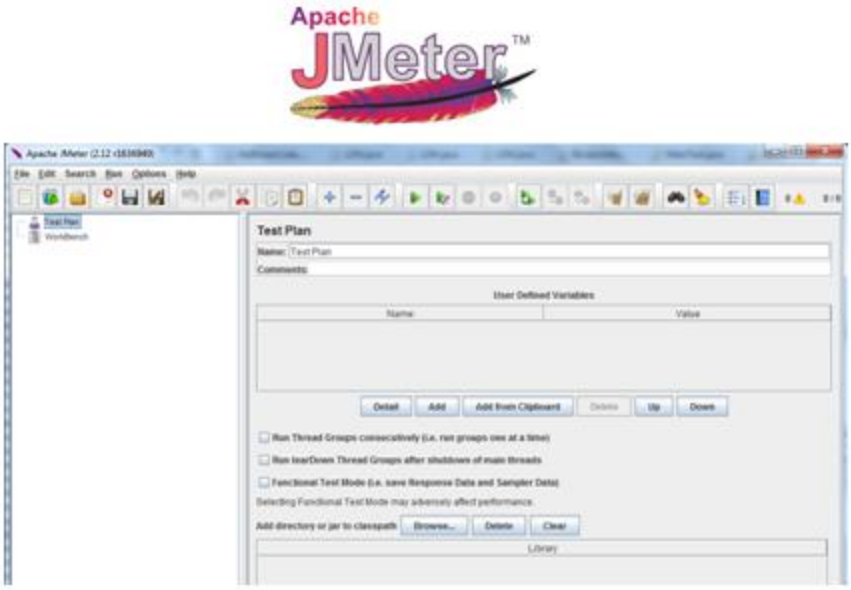

Figura 13: Aplicación Apache JMeter

2.5 Big Data

El término Big Data se refiere a colecciones de datos que son tan grandes o complejos que no se pueden tratar como conjuntos de datos tradicionales. Con el avance de la tecnología, datos de comunicación entre máquinas, datos producidos por los usuarios en la web 2.0, datos acumulados por diversas industrias o centros de investigación, ha surgido una nueva problemática, ya que no solo está el problema del gran volumen de datos, sino que también estos datos pueden ser extremadamente variados entre sí, y la construcción de aplicaciones que analicen estos datos a una velocidad adecuada, o los capturen, busquen, compartan o almacenen no se pueden realizar con tecnologías tradicionales como las bases de datos convencionales [40].

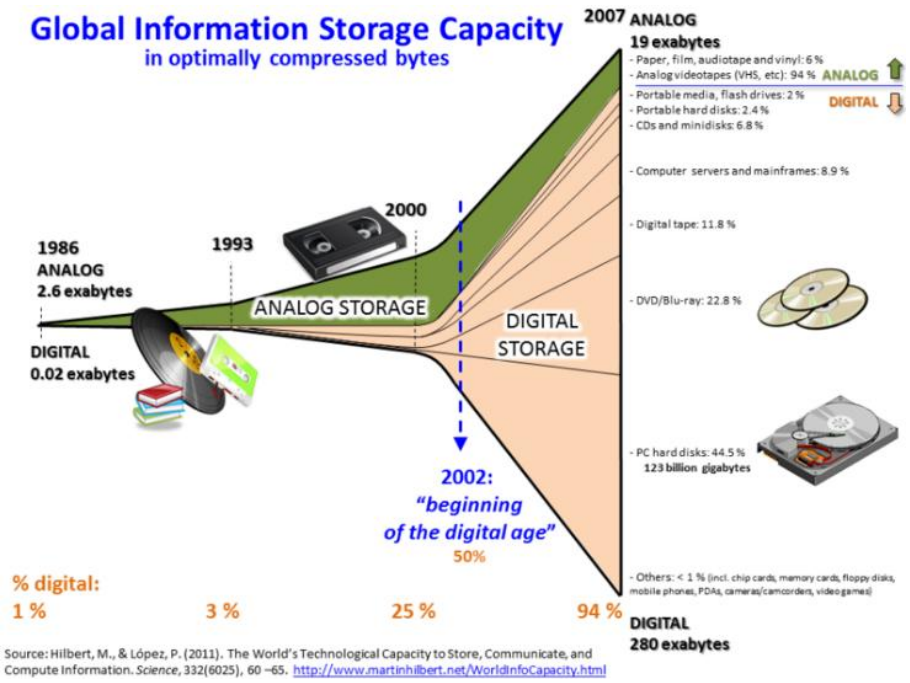

Figura 14: Crecimiento de digitalización y capacidad global de almacenamiento de la información

2.5.1 Bases de datos NOSQL

Las bases de datos NoSQL (Not Only SQL) son sistemas de almacenamiento de información que no cumplen con el esquema entidad-relación, están orientadas para trabajar con una gran escalabilidad para manejar volúmenes de datos de trabajo con Big Data.

Respecto a las bases de datos tradicionales se diferencian en que las bases de datos NOSQL no basan su funcionamiento en tablas, joins y transacciones, si no que proveen un esquema más flexible [41]. En función de este esquema se distinguen cuatro tipos fundamentales de bases de datos NOSQL:

- Almacenamiento clave-valor: Se accede a los datos a partir de una clave única [42]. Al no ofrecer una estructura de datos claves son de mayor utilidad en operaciones simples basadas en claves.
- Almacenamiento documental: Parecidas a las de almacenamiento clave-valor, pero este tipo de bases de datos sí que requiere una estructura de datos concreta [43], a estos datos se les llama documentos, y pueden tomar distintos formatos como JSON o XML. Además permite lanzar queries sobre estos datos [42]. MongoDB o SimpleDB son las tecnologías de este tipo más usadas. Y las tecnologías MapReduce de google y Hadoop de apache también están basadas en este concepto.

- Almacenamiento en grafo: No se basan en tablas sino en grafos, almacenando la información en los nodos mientras que las relaciones las representan las aristas [43]. Las operaciones que antes eran entre tablas ahora son recorridos del grafo, actualizando siempre una lista de adyacencia entre los nodos [42]. GraphDB es la tecnología más utilizada en este aspecto.
- Almacenamiento orientado a columnas: Este tipo de almacenamiento es parecido al Documental. Su modelo de datos es definido como "un mapa de datos multidimensional poco denso, distribuido y persistente" [42]. Está orientado al almacenaje de datos que escalan horizontalmente, permitiendo que bajo una misma clave se almacenen varios tipos de atributos. HBase o HyperTable son las principales tecnologías que implementan este tipo de base de datos.

2.5.2 Minería de datos

Data mining o minería de datos en un subcampo interdisciplinar de las ciencias de computación cuyas principales tareas se pueden dividir en dos: predicción, que consiste en hacer uso de los valores conocidos para determinar valores de los datos desconocido o futuros, y descripción que consiste en el descubrimiento de patrones sobre conjuntos de datos muy grandes (big data), para extraer información estructurada de forma comprensible [44]. Engloba aspectos de la inteligencia artificial, aprendizaje automático, estadística y sistemas de bases de datos [45].

Los principales patrones que se reconocen gracias a la minería de datos y hasta ahora desconocidos son los grupos de registros de datos en el que se centra el análisis de clusters, los registros poco usuales en los que se centra la detección de anomalías y el estudio de dependencias en el que se basa la minería por reglas de asociación.

2.5.2.1 Técnicas de minería de datos

Algunas de las principales técnicas de minería de datos, basadas en el descubrimiento de información entre los datos son [44]:

- Reglas de asociación: Se usan para descubrir fenómenos en común que ocurren en un conjunto de datos.
- Reglas de clasificación: Basado en el descubrimiento de reglas que permiten particionar los datos en conjuntos disjuntos.
- Redes neuronales: Inspirado en la forma en que funciona el sistema nervioso, se construye un sistema de aprendizaje y procesamiento automático en el que un sistema de neuronas compone una red que colaboran entre ellas para producir la salida.

- Clustering: Es un proceso de agrupamiento de datos dentro del conjunto, basado en distintos criterios, siendo el más popular la distancia entre estos datos, dando lugar a conjuntos de datos similares dentro del conjunto de datos inicial. En la Figura 15 vemos un ejemplo de este proceso.

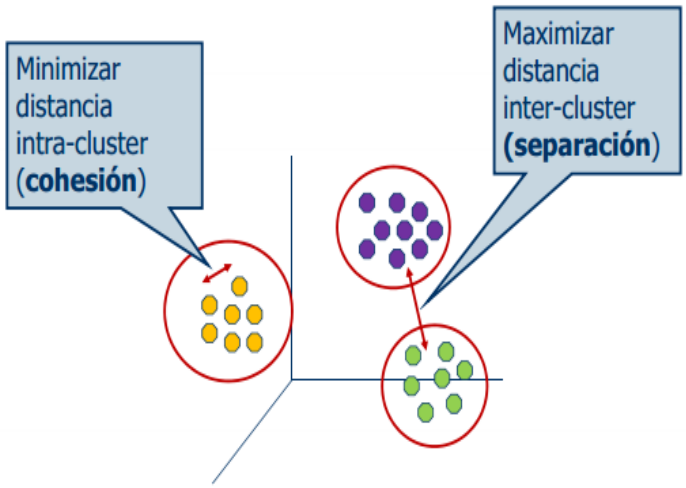

Figura 15: Clustering de un conjunto de datos

2.5.2.2 Técnicas de clusternig.

En este trabajo se han usado técnicas de clustering para realizar minería de datos, en esta sección describimos las principales.

Clustering basado en densidad con el algoritmo DBScan

DBScan (el agrupamiento espacial basado en densidad de aplicaciones con ruido) es un algoritmo de clustering basado en la densidad de los datos y diseñado para descubrir clusters de formas arbitrarias [46]. Es muy útil cuando los datos se distribuyen en zonas de densidad alta y baja. En la Figura 16 vemos un ejemplo de esta técnica de clustering.

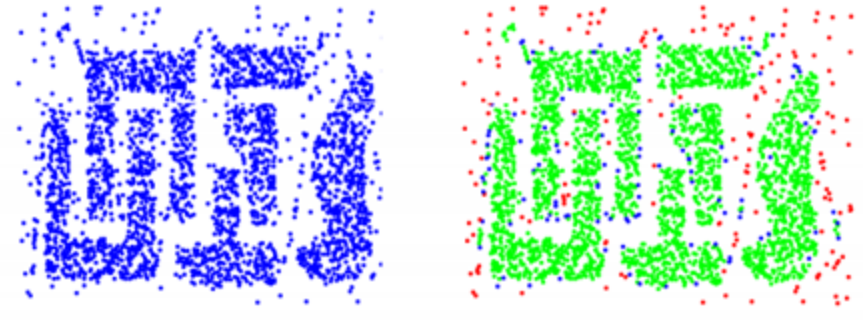

Figura 16: Clustering con DBScan

Clustering basado en ordenamiento jerárquico

El ordenamiento jerárquico es un método de clustering basado en la construcción de una jerarquía de clusters [47], esto se realiza mediante la construcción de un árbol donde las hojas son los elementos del conjunto y el resto de nodos son subconjuntos que se pueden particionar a su vez resolviendo así el problema de determinar el número de conjuntos (clusters) en los que se dividirá el conjunto de datos ya que se calcula de forma automática, como se ve en la Figura 17.

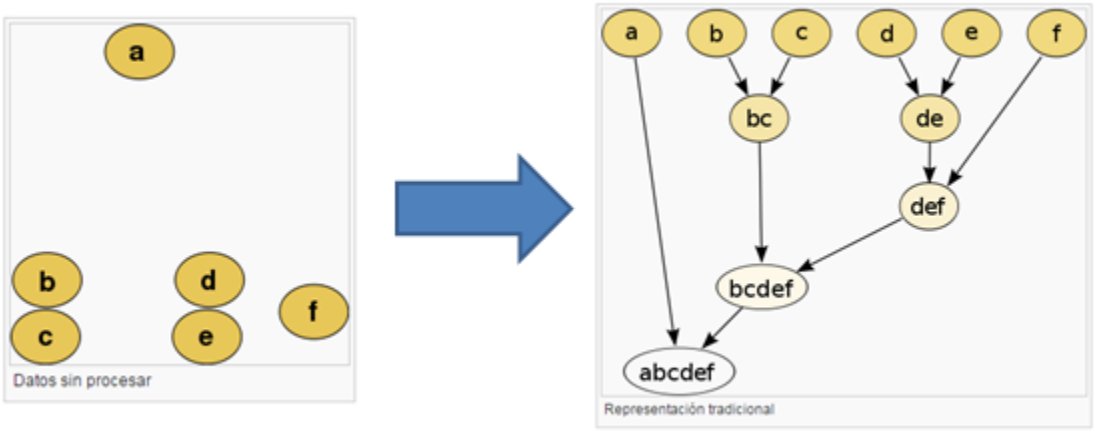

Figura 17: Representación del clustering mediante el ordenamiento jerárquico

El coste de los algoritmos de esta técnica de clustering es muy elevado ($O \geq n^2$) [48] [49]por lo que no son algoritmos escalables.

Clustering basado en ordenamiento no jerárquico con el algoritmo K-means

K-means es un algoritmo de clustering que consiste en la partición de un conjunto de datos en k clusters en los que cada dato pertenece al grupo cuya media sea más cercana a su valor, este valor de la media está representado por el centroide del cluster.

El problema de clustering usando k-means con espacios de d dimensiones es un problema NP-difícil [50]. Pero se fija el número de clusters k, y el número de dimensiones d, la complejidad es polinómica respecto al número de datos a clasificar (n) [51]:

$$\mathrm{O}(n^{dk+1} \log n)$$

El algoritmo comienza colocando k puntos en el espacio representado por los objetos que se van a agrupar, representando el grupo inicial de centroides.

Se asigna cada objeto al grupo que tiene el centroide más cercano, con una función distancia entre objetos (distancia Manhattan, distancia Minkowski, por ejemplo).

Cuando todos los objetos han sido asignados, se recalculan las posiciones de los centroides, esto se repite hasta que el grupo de centroides no varíe [52].

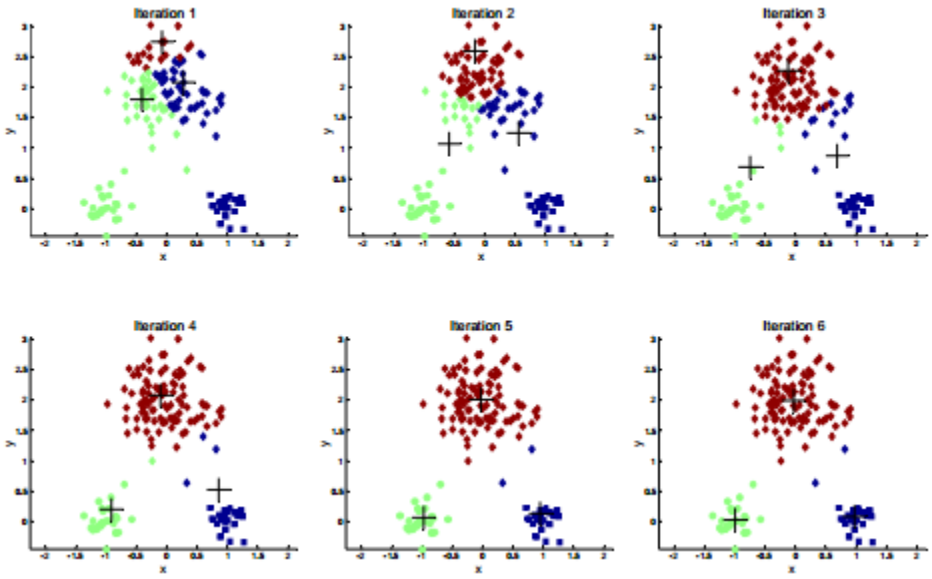

Figura 18: Aplicación del algoritmo K-Means

La métrica SSE (Sum of Squared Error) se usa para ayudar a decidir el número de clusters en que dividiremos el conjunto de datos al usar el algoritmo k-means.

La métrica mide la suma del cuadrado de las distancias entre cada miembro del cluster y su centroide según la siguiente fórmula.

$$SSE = \sum_{i=1}^{k} \sum_{x \epsilon C_i} d^2(m_i, x)$$

Donde m_i es el centroide de cada cluster, k el número de clusters y Ci cada uno de los clusters.

Esta métrica mide la cohesión interna del cluster y la separación externa, teniendo su valor optimo cuando deja de decrecer abruptamente [53]. Se usa para validar el valor de k.

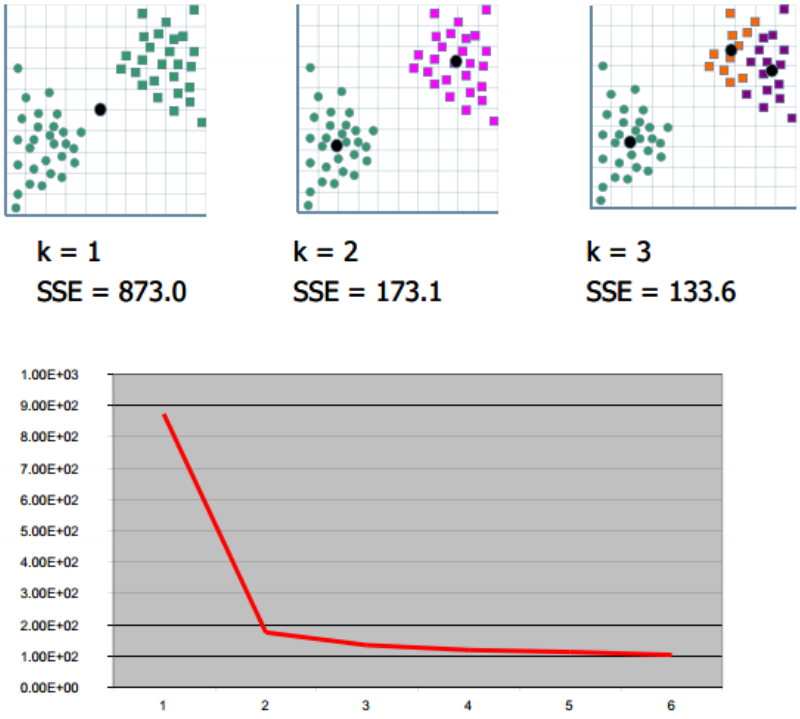

Figura 19: Ejemplo de validación de k usando la métrica SSE

2.6 Lematización

La lematizacion es el proceso por el que se reduce una palabra a su raíz o lema, es usado en el campo de la morfología lingüística y la recuperación de información. Este lema al que se reducen las palabras no siempre tiene por qué coincidir con su raíz morfológica [54].

El primer algoritmo de lematizacion fue publicado en 1968 por Julie Beth Lovins [54], pero fue en 1980 cuando Martin Porter desarrollo en principal algoritmo de lematización en el que se basan los algoritmos usados hoy en día [55].

2.6.1 Algoritmos de lematización

En esta sección se describen las principales formas de implementar un algoritmo lematizador en función de la precisión, rendimiento y resolución de problemas que puedan surgir:

- La técnica de producción: Este algoritmo se basa en el uso de una tabla que contiene todos los lemas y para cada palabra se busca en la tabla para obtener el mismo, aunque se pueden devolver varias formas.

- Algoritmos de desmontado de sufijos: Estos algoritmos constan de reglas almacenadas que ayuda a recortar la palabra de entrada hasta encontrar su lema.

- Algoritmos estocásticos: Estos algoritmos incluyen el uso de la probabilidad para identificar el lema de una palabra, aprenden de una tabla con las formas de las raíces hasta sus declinaciones para desarrollar un modelo probabilístico que se expresa en forma de reglas lingüísticas similares a las de los algoritmos de desmontado de sufijos.

2.6.2 Lematización en Español

El estudio de la lematización está centrado en su mayoría sobre el inglés, pero también se han adaptado los algoritmos al español, en especial gracias al trabajo de Martin Porter que creó el lenguaje de programación Snowball [56], diseñado para crear algoritmos lematizadores que se usen en los casos de recuperación de la información, y gracias a esto se ha permitido la construcción de algoritmos lematizadores en otros idiomas como el francés, italiano y español [57].

2.7 OpenId Connect

OpenID Connect es una capa de autenticación de identidad sobre el protocolo OAuth 2.0 [58], que permite la autenticación delegada de un usuario en una aplicación delegando la verificación de identidad por un servidor de autorización bien conocido y de confianza, como se describe en la Figura 20.

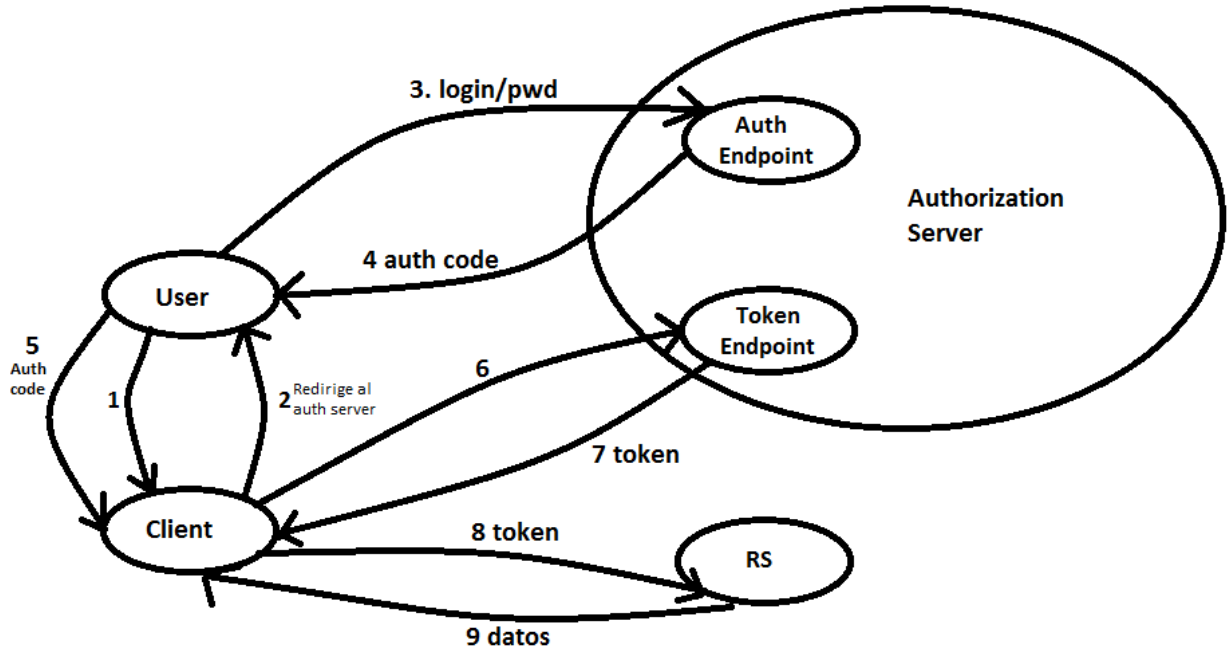

Figura 20: Flujo de autorización de OAuth 2.0

El protocolo OAuth 2.0 define estos flujos de autorización [59], pero para realizar la autenticación delegada se usa el estándar OpenId Connect, que sobre OAuth define la estructura del token de identidad y el *scope* (ámbito) del token para establecer que datos requeridos del usuario.

$$OIDC = OAuth\ 2.0 + IdToken + Scopes$$

El protocolo OAuth se empezó a desarrollar en Noviembre de 2006, pero fue en abril de 2007 cuando DeWitt Clinton de Google se interesó en apoyar el proyecto en el que en diciembre de ese mismo año se presentaba el borrador definitivo [60].

La versión 1.0 del protocolo se presentó en Abril de 2010, y en Octubre de 2012 se presentó la versión 2.0 que se usa en este proyecto.

2.7.1 OpenID sobre Oauth

Cuando se usa OpenID, el *authorization server* es un proveedor de identidad, como puede ser Google, Facebook o Amazon, contra los que el usuario se autentica y te devuelven un code que servirá para verificar esa autenticación por el servidor de la aplicación contra la que se quiera autenticar.

En la Figura 21 se describe el flujo de esta operación usando Google como el proveedor de identidad.

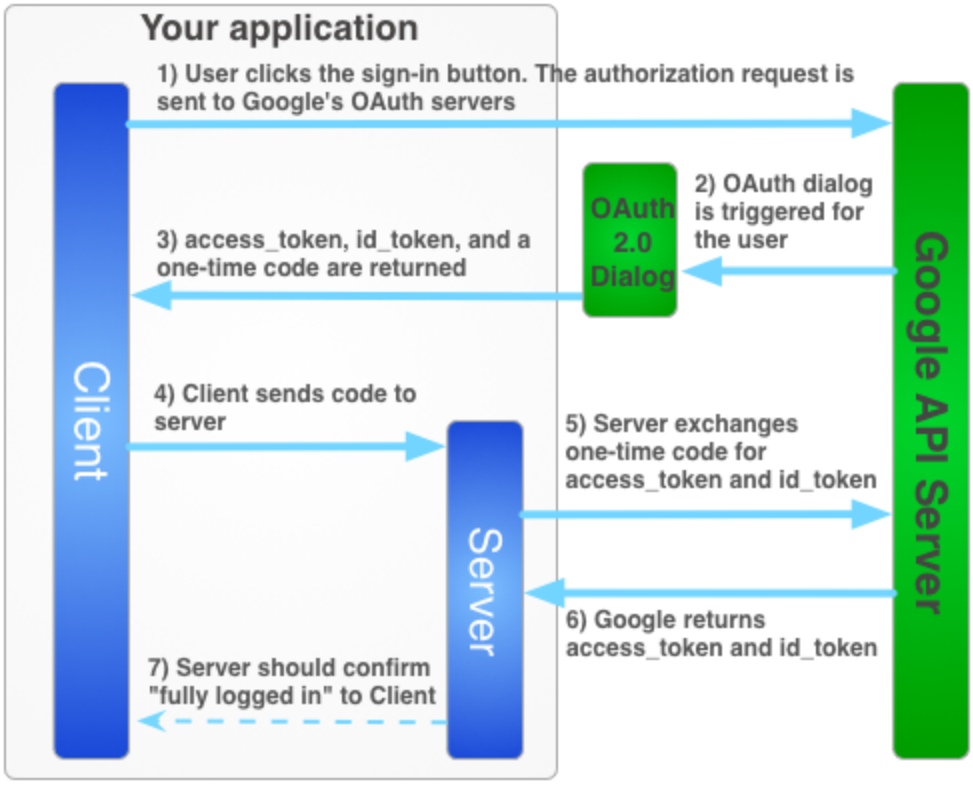

Figura 21: Flujo de autenticación con OpenId sobre Google

3 Módulo Resource Sharing Center

El Resource Sharing Center (RSC) es un servicio que permite a los usuarios subir y compartir recursos tanto lógicos (ficheros) como físicos (servicios) a un repositorio en la nube utilizando como mecanismo de compartición el concepto de círculo (amigos, compañeros, etc.…), que es la forma natural de compartir utilizada en redes sociales.

Figura 22: Resource Sharing Center

Se diferencia respecto a otros servicios similares como Dropbox, Oboom, etc.… En que el Resource Sharing Center incorpora la posibilidad de buscar recursos compartidos por otros usuarios e incorporarlos en el inventario de recursos del usuario, el cual se divide en recursos propios y recursos de terceros.

Otra novedad es el motor de recomendación, en base a la interacción del usuario con la aplicación, y los recursos que comparte, y los intereses que decida incluir, se generará un perfil del usuario, que permitirá agrupar a los usuarios en clusters con otros usuarios con perfiles similares y hacer recomendaciones sobre recursos que puedan interesarte.

El acceso a la aplicación se puede hacer mediante login directo o usando autenticación delegada siguiendo el protocolo open id y usando la cuenta propia de google, como podemos ver en la Figura 23.

Figura 23: Página de login del RSC

El servicio sigue la arquitectura cliente servidor, estando el servidor desarrollado en dos módulos, uno Python y el otro en Node.js aprovechando las ventajas de la ejecución mono hilo y la función de callback. El servidor Python se encarga de mostrar el front-end, desarrollado en HTML5 e interactuar con el motor de recomendaciones. En la Figura 24 podemos ver este despliegue.

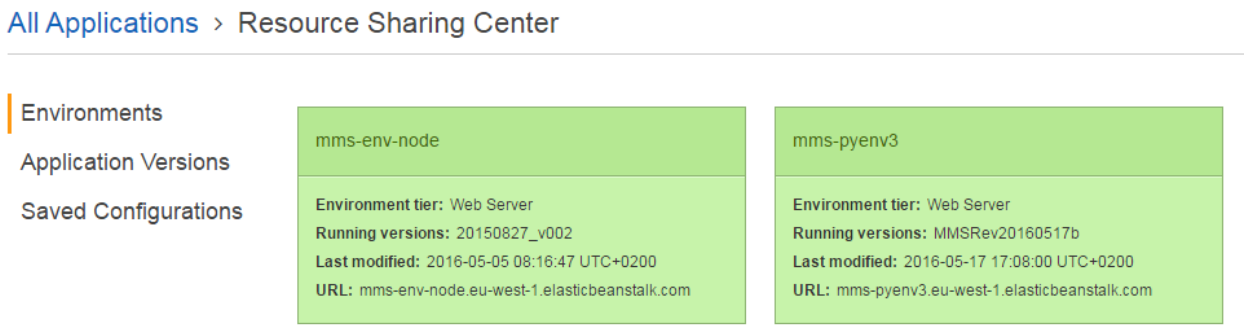

Figura 24: Despliegue del RSC en Elastic Beanstalk

3.1 Arquitectura y modelo de datos

En este apartado de la memoria, se explicara el diseño del servicio Resource Sharing Center, y más específicamente la descripción de su funcionalidad, los principios de diseño que han guiado el desarrollo y el modelo de datos de las bases de datos.

3.1.1 Principios de diseño

Esta sección describe los principios de diseño básicos de cara a optimizar recursos en términos de tráfico, computación y costes.

3.1.1.1 Desarrollo en AWS

En esta sección se describen específicamente los principios de diseño orientados al ahorro de recursos y costes durante la interacción del back-end con las herramientas de Amazon AWS

- Operaciones no bloqueantes en la interacción entre los distintos módulos de AWS (acceso a SimpleDB, S3, etc..) para conseguir:
    - Maximizar la concurrencia reduciendo costes al realizar un mayor número de llamadas a una instancia.
    - Mejorar la experiencia de usuario reduciendo los tiempos de respuesta.
- Evitar el tráfico saliente del servidor, siempre que sea posible es preferible el trafico entrante en los servicios de AWS esto es, que el cliente se comunique directamente con el servicio que necesita:
    - El trafico entrante en los servicios de AWS es gratis
    - El trafico saliente del servidor alojado en Beanstalk aumenta costes y reduce la concurrencia del mismo generando costes computacionales extra.

Estos principios aplicados al diseño del RSC dan como resultado el diseño de alto nivel que se muestra en la Figura 25.

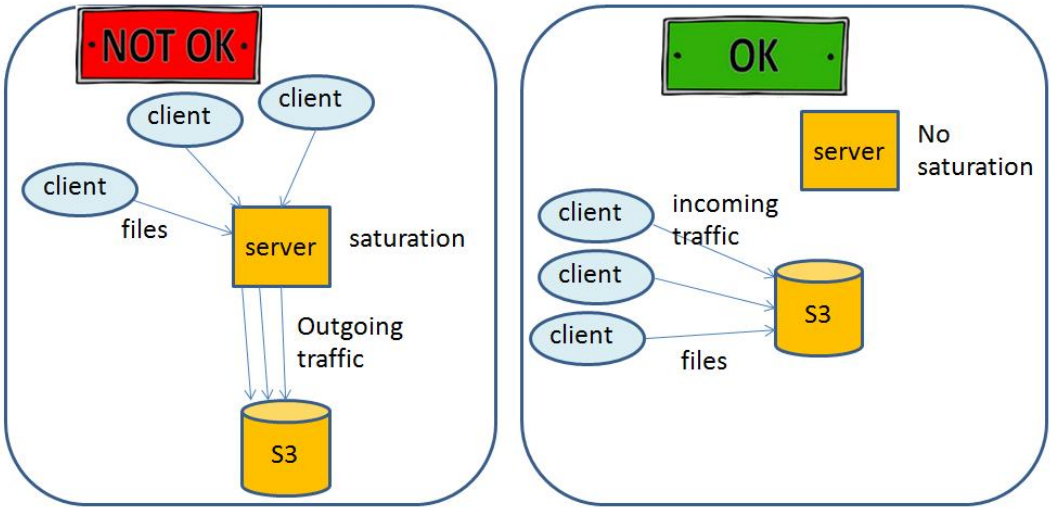

Figura 25: Principio de diseño de desarrollo en AWS

3.1.1.2 Peticiones fuera de dominio (CORS)

En esta sección se comentan los principios de diseño que se siguen para desarrollar la aplicación teniendo en cuenta la seguridad respecto a las peticiones de fuentes en otro dominio (Cross Origin Resource Sharing).

En los desarrollos con Amazon, si se quieren hacer pruebas en local sin subir el código a Amazon, por ejemplo, una página web en HTML5 con JavaScript que invoque a Amazon y dicha página está siendo cargada desde el disco duro, su origen será el disco duro. Si dicha página de forma autónoma (AJAX) hace peticiones a Amazon, estas fallarán, porque el browser detectará que la página está haciendo peticiones fuera de su dominio origen sin consentimiento explícito.

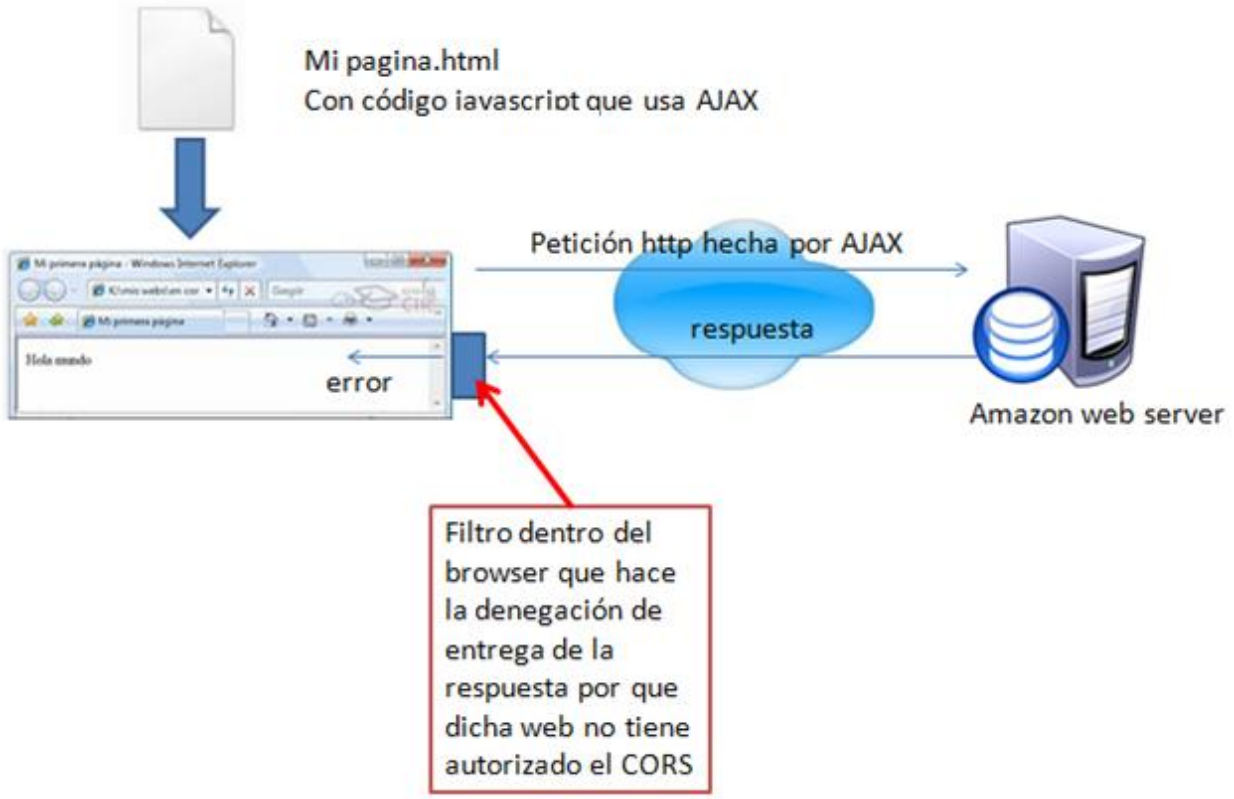

Figura 26: Error en la petición desde distinto origen

Este tipo de seguridad está implementada en los navegadores para evitar que un usuario que visita una página maliciosa cargue un script malicioso (en JavaScript por ejemplo) y dicho script acceda a lugares donde no puede entrar debido a un cortafuegos pero a través de una víctima que cargue el script ya puede acceder, como se ve en la Figura 27:

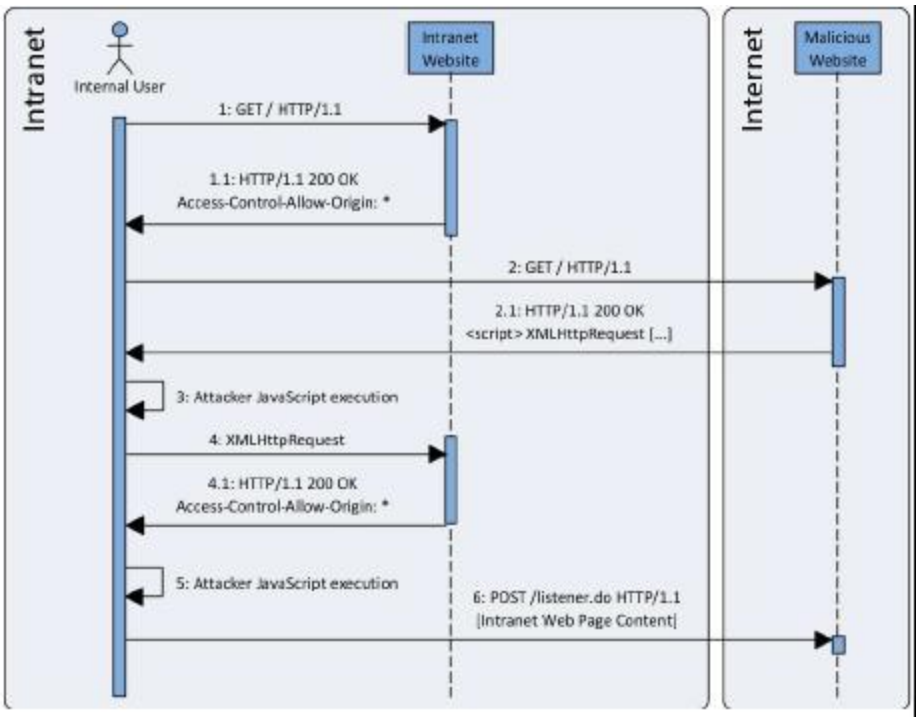

Figura 27: Ataque desde un origen distinto al de la aplicación

Para poder autorizar que la respuesta de una página que reside en un web server sea procesada en el browser y entregada al script que la invoca, se deben enviar desde el servidor web las cabeceras:

Access-Control-Allow-Origin: http://www.dominio-tercero.com

O bien

Access-Control-Allow-Origin: *

Con esto último se permite que se acceda desde cualquier dominio origen

Además se debe hacer que el browser vea que el servidor acepta que le hayan hecho una petición usando AJAX, de modo que también hay enviar esta cabecera en la respuesta:

Access-Control-Allow-Headers: X-Requested-With

Si no se envía eso, el browser pensará que el servidor web no quiere que le hagan peticiones Ajax y bloqueará la respuesta, es decir, la respuesta llegará al browser pero al no encontrarse esa cabecera, no será procesada.

Este principio de diseño también se aplica dentro de AWS donde también tenemos que manipular un fichero CORS para permitir que las aplicaciones accedan a otros servicios de S3, como se ve en la Figura 28.

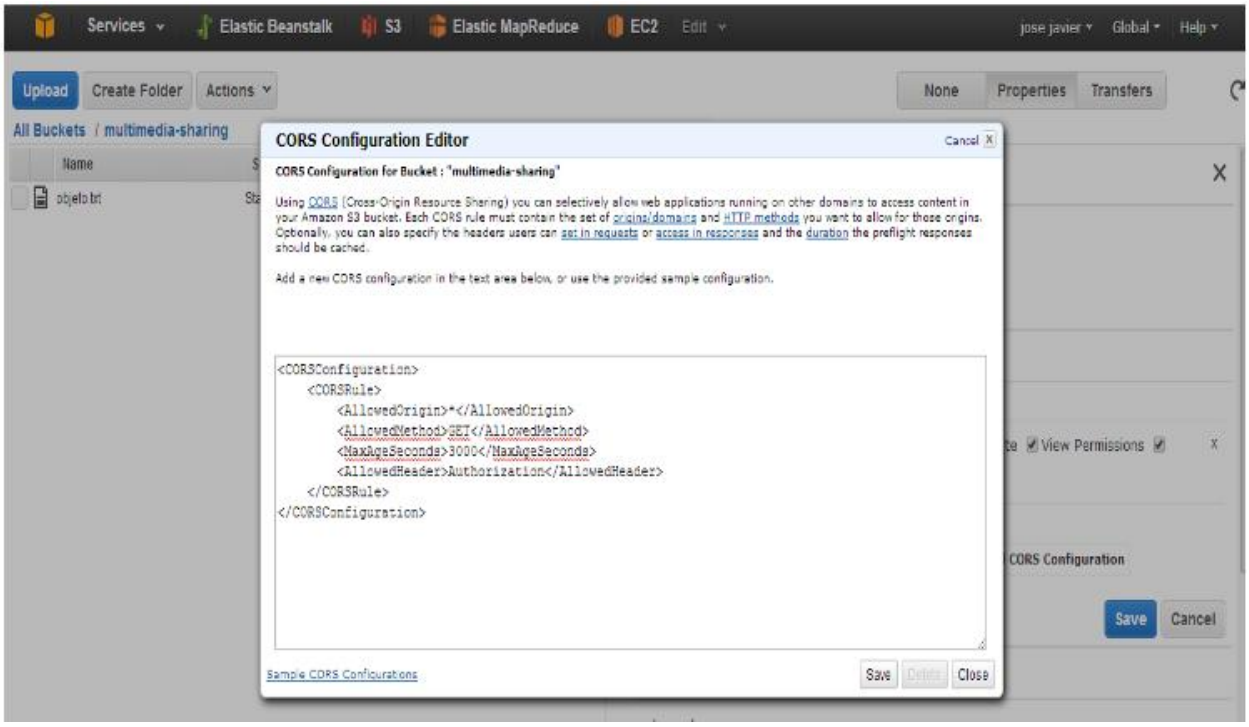

Figura 28: Fichero CORS en AWS

3.1.1.3 Comunicación uniforme

Para implementar de forma uniforme y robusta la comunicación front-end-back-end, y que permita múltiples implementaciones de las aplicaciones en modo cliente, se establecen las respuestas en formato JSON con una forma definida.

3.1.1.3.1 Formato de las respuestas JSON

Las respuestas son objetos JSON con los siguientes atributos.

- Result: “SUCCESS” | “FAILURE”
- Code: <Valor numérico>
- Description : <Texto descriptivo opcional>
- Data: <objeto JSON>

Ejemplos:

- GetProfile()

```
{"result":"SUCCESS",
"code":0,
"description":"operation ok",
"data":[
```

```
{"key":"CASA","weight":1},
{"key":"PERRO","weight":1}
]}
```


- getSharedResources()


```
{"result":"SUCCESS",
"code":0,
"description":4,
"data":[
{"KEYWORDS":["CASA","PERRO”],
"CIRCLES":[{"circle_id":"PUBLIC","circle_name":"PUBLIC"}],
"DESCRIPTION":"cccc","VALUE":"url2","TYPE":"FILE"},
{"KEYWORDS":["CASA","PERRO"],
"CIRCLES":[{"circle_id":"PUBLIC","circle_name":"PUBLIC"}],
"DESCRIPTION":"dedede","VALUE":"url4","TYPE":"FILE"},
{"KEYWORDS":["CASA","PERRO"],"CIRCLES":"ALL","DESCRIPTION":"dddd","VALUE":"ur
l3","TYPE":"FILE"},
{"KEYWORDS":["CASA","PERRO"],
"CIRCLES":[{"circle_id":"PUBLIC","circle_name":"PUBLIC"}],
"DESCRIPTION":"dede ded","VALUE":"url7","TYPE":"FILE"}]}
```


3.1.2 Componentes

El RSC está compuesto por dos módulos principales, basados en la arquitectura cliente y servidor. En el módulo cliente está la página del servicio desarrollada en HTML5 con Twitter Bootstrap, y el módulo servidor está compuesto por dos submódulos, el módulo “Sharing” implementado en Node.js y el módulo “Hello”, que esta implementado en Python.

Por último y para realizar funciones muy específicas, hay un módulo “Batch”, que se ejecuta periódicamente y que tiene funciones para generar la tabla de índice inverso, y el clustering que se definirá en el módulo Analytics. En la Figura 29 se muestra esta descripción de la arquitectura.

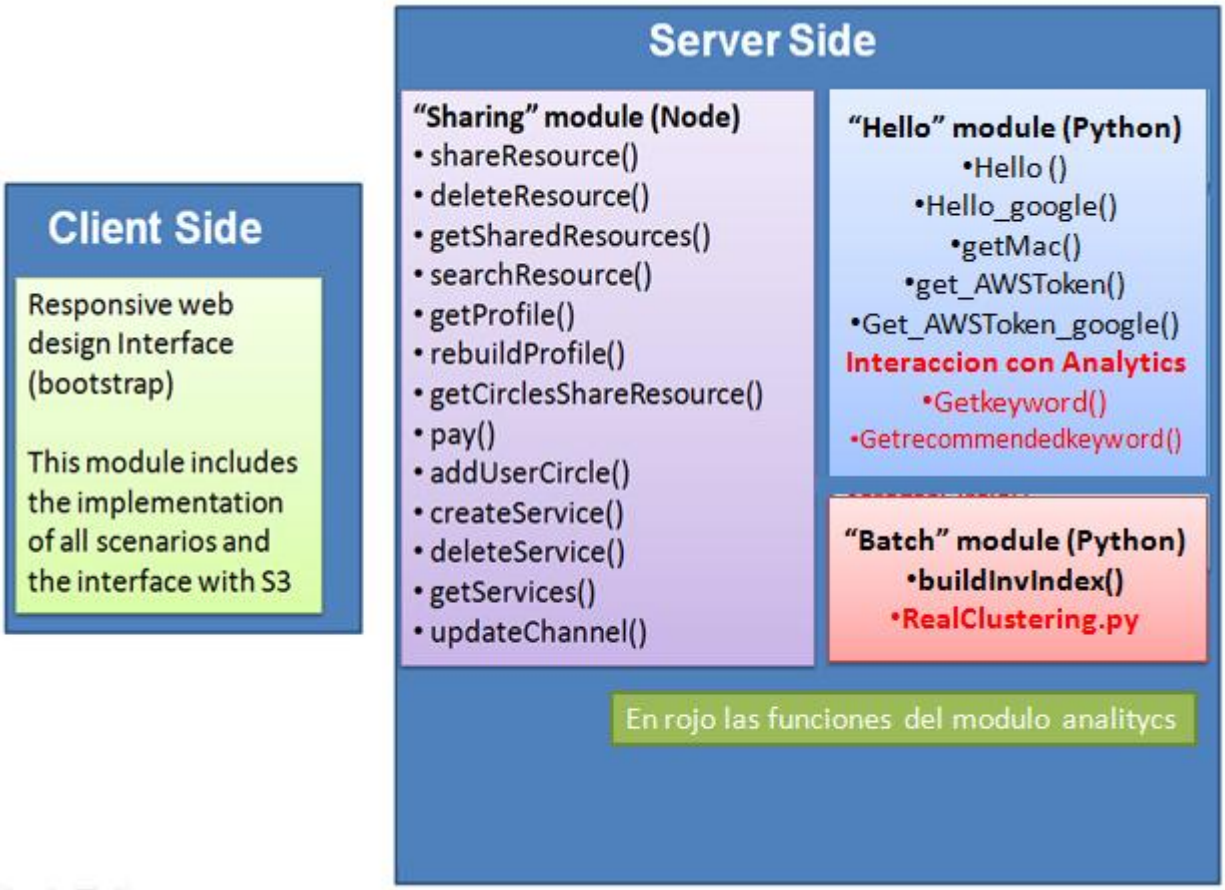

Figura 29: Arquitectura del servicio

3.1.2.1 Lado servidor

Esta desplegado en Elastic Beanstalk como una sola aplicación con dos entornos de AWS.

Un entorno en Node.js (la maquina principal que se instancia en EC2 y todas las maquinas que se podrán instanciar en función a las llamadas que se realicen, cargarán el código definido en este entorno), aprovechando las ventajas del *callback* y la *ejecución monohilo* para realizar la comunicación con el back-end, ya que puede enviar varias peticiones y seguir ejecutando otras funciones.

El otro entorno en Python se encarga de las funciones que necesitan una respuesta para poder continuar con su ejecución, como la autenticación de usuario y creación de su carpeta por defecto, y la comunicación con el módulo Analitycs.

3.1.2.2 Lado cliente

La página de la aplicación la desplegará el módulo Hello, es una página usando HTML5, JavaScript para las funciones de la página, AJAX sobre esto, para la comunicación con el back-end, más específicamente, con las funciones del módulo Sharing, y Twitter Bootstrap para el diseño visual de la página y *responsive web design*.

El diseño del lado cliente se puede dividir en tres módulos, el módulo de interacción con el back-end, compuesto por las funciones que contienen las llamadas AJAX, el módulo de interacción con la página, que contiene las funciones que se ejecutan sobre la página, y el módulo de diseño, con el diseño en Bootstrap de la página.

3.1.3 Modelo de datos

En esta sección se describirán los tipos de datos que hay en las bases de datos, la forma en que se relacionan y las condiciones que deben cumplir los mismos.

3.1.3.1 Dominio MMS_RESOURCE_TABLE

En este dominio incluimos los objetos compartidos en el RSC:

Tipo de campo	Nombre del Campo	Comentario
Nombre del objeto	USER_ID+VALUE	
Atributo	USER_ID	
Atributo	TYPE	
Atributo	VALUE	Url del recurso
Atributo	LIST_OF_CIRCLES	Lista de círculos donde es compartido
Atributo	DESCRIPTION	Texto
Atributo	KEYWORDS	Lista de palabras clave extraídas automáticamente del texto de la descripción.
Atributo	TIMESTAMP	Cuando fue creado o actualizado.

Atributo	PICKUP_COUNTER	Se incrementa cuando otro usuario realiza la función pickup() sobre el recurso para añadirlo a su inventario de terceros.
Atributo	OWNER	Dueño del recurso.
Atributo	OWNED	YES cuando USER_ID=OWNER
Atributo	SUBSCRIPTION_ID	Solo si el usuario está suscrito a este recurso.
Atributo	SERVICE_NAME	Solo si el recurso es de tipo Servicio
Atributo	PRICE	Solo si el valor del círculo empieza por: "MARKET-$$$$$$$"

Donde:

TYPE = "FOLDER" | "SERVICE" | "INTEREST" | "FAVOURITE_LINK"

Y VALUE toma los siguientes valores, dependiendo de TYPE:

- TYPE= "FOLDER", entonces el VALUE es la url de S3.
- TYPE= "SERVICIO", entonces el VALUE es la url del servicio
- TYPE="INTEREST" entonces el VALUE es "INTEREST"
- TYPE="FAVOURITE_LINK" entonces el VALUE es la url del link.

La lista del atributo KEYWORDS es de la siguiente forma: ["MUSIC","SHAKIRA", "POP"]

La razón para tener dos campos tan similares como Owner y Owned es que SimpleDB no permite hacer búsquedas del tipo:

select * from MMS_RESOURCE_TABLE where USER_ID=OWNER_ID

El término a la derecha de la igualdad debe ser una constante, por eso se usa el campo OWNED.

3.1.3.2 Dominio USER_PROFILE_TABLE

En este dominio se guarda la información sobre los usuarios, y sus círculos y carpetas por defecto.

Tipo de campo	Nombre del campo	Comentario
Nombre del objeto	Valor de USER_ID	
Atributo	DEFAULT_BUCKET_NAME	Carpeta por defecto de cada usuario
Atributo	PROFILE	
Atributo	OTHER_CIRCLES	Círculos de otros usuarios a los que pertenece.

El perfil esta en formato JSON con la siguiente forma:

[{"key":"MUSIC","weight":2},

{"key":"SHAKIRA","weight":1},

{"key":"LENNON","weight":1},

{"key":"COMPUTERS","weight":1}]

3.1.3.3 Dominio SERVICE_PROFILE_TABLE

En este dominio se almacenan los servicios (recursos físicos) que han dado de alta los usuarios

Tipo de campo	Nombre del campo	Comentario
Nombre del objeto	Valor de SERVICE_ID	(user_id+service_name)
Atributo	SERVICE_NAME	Nombre del servicio
Atributo	USER_ID	Dueño del servicio
Atributo	DESCRIPTION	Descripción del servicio
Atributo	KEYWORDS	Lista de palabras clave extraídas automáticamente

		del texto de la descripción.
Atributo	SERVICE_URL	Url del servicio
Atributo	FUNCTIONAL_PARAMS	Parámetros funcionales, definidos con el servicio.
Atributo	NON_FUNCTIONAL_PARAMS	Parámetros no funcionales, definidos con el servicio.

3.2 Diseño funcional servidor

En este apartado se realizara una descripción funcional del servidor, especificando las funcionalidades del servicio.

3.2.1 Funcionalidades

El Resource Sharing Center tiene las siguientes funcionalidades:

- Autenticación delegada: Puedes acceder al servicio con tu cuenta de google, ya que AWS permite la autenticación delegada basada en el protocolo OpenId.

- Acceso temporal con token de AWS: Al loguearte en la aplicación se inicia un dialogo con AWS para pedir un token temporal que te dará acceso a los servicios de AWS que la aplicación utiliza.

- Creación de la carpeta en S3 por defecto:
  La carpeta por defecto será la carpeta raíz del usuario, si el usuario ya tiene una carpeta raíz el RSC le devolverá el nombre de esta en la tabla USER_PROFILE_TABLE. Si no tiene, el RSC creará una carpeta raíz para el usuario asignándola un nombre único para cada carpeta raíz de usuario.

  La carpeta por defecto es creada en el lado cliente, pero el nombre para esta te lo da el lado servidor. Esta estrategia es robusta y reduce costes, ya que no envía tráfico saliente del servidor a S3.

- Creación y borrado de carpetas:
  Estas funciones están implementadas en el lado cliente, que invoca al servicio S3, y dentro de la carpeta raíz del usuario permite crear un árbol de carpetas para organizar los recursos lógicos de los que dispone.

- Subida de contenido multimedia:
  Diálogo entre la aplicación cliente con S3 sin la interacción del lado servidor.

- Compartición de recursos (contenido multimedia, links favoritos y servicios):
  Los usuarios pueden compartir objetos invocando la función en el lado servidor del RSC, que actualiza la tabla MMS_RESOURCE_TABLE indicando si un objeto particular es compartido y en que círculos sociales.

  Un usuario puede compartir servicios, carpetas (no ficheros por separado), y links favoritos que son compartidos y públicos por defecto.

  Cuando un usuario comparte un objeto, tiene que rellenar un campo de descripción, sobre el que se aplicará un algoritmo de lematización para obtener las palabras clave e implementar la función de búsqueda sobre estas.

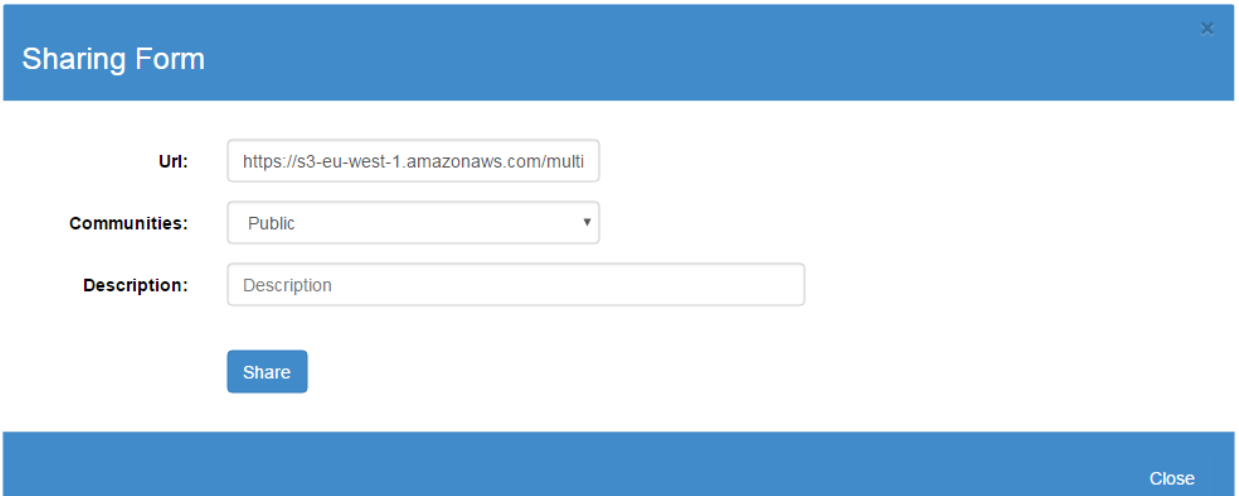

Figura 30: Ventana modal con formulario de compartición

- Funcionalidad Marketplace:
  Los usuarios pueden elegir un precio para los contenidos que comparten, para hacer esto, tienen que compartir el contenido en el círculo "MARKET" y poner un precio, en este caso el acceso al círculo social se hace cuando el usuario paga por el contenido.
  De cara al usuario esto es transparente, pero el círculo MARKET para cada contenido de pago tendrá un nombre distinto que se genera aleatoriamente para que un usuario que pague por un contenido no tenga acceso a todos los contenidos de pago del dueño.

- Búsqueda de contenido multimedia, links favoritos y servicios.
  Cada vez que un usuario comparte un objeto, es obligatorio establecer metadatos, las palabras clave que el algoritmo lematizador obtiene de la descripción. Así, es posible buscar y encontrar estos objetos compartidos usando búsquedas en la base de datos basadas en esas palabras clave.

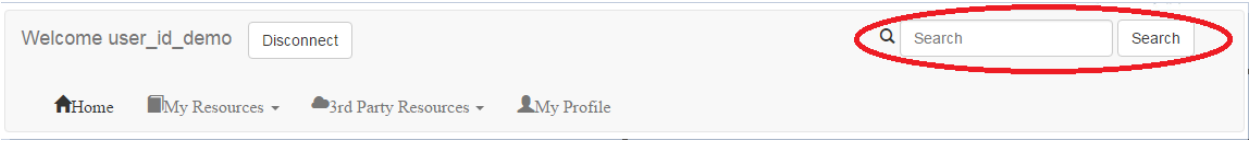

Figura 31: Formulario de búsqueda en barra de navegación

- Generación de thumbnails:
  En el lado cliente, cada vez que un usuario suba una imagen se generará un thumbnail que es el que se verá desde la aplicación cliente, para no tener que cargar siempre la imagen original que puede ser muy pesada en el navegador.
  Esta función es transparente al usuario que no podrá interactuar con los thumbnails, solo los ve en el navegador.

- Motor de recomendación basado en el perfil de usuario:
  Esto es parte del otro módulo de la aplicación, el módulo Analytics. Basándose en el perfil del usuario, se ofrecerá una recomendación al usuario de un recurso que le pueda interesar.

3.2.1.1 Autenticación y acceso

Una vez el usuario se loguea, el RSC le devuelve un *token temporal de AWS*, este token permitirá durante un tiempo limitado que el usuario interactúe con los servicios de AWS (S3 y SimpleDB, almacenamiento y base de datos)

El *token AWS* es generado en el módulo servidor en Python invocando la función assume_role() o asume_role_with_web_identity() (dependiendo de si el acceso es directo a la aplicación o mediante un proveedor de identidad) perteneciente al API de gestión de identidad y acceso (AWS IAM) de AWS y es enviado a la parte cliente junto con la respuesta al login.

Este token provee de acceso limitado a los recursos de Amazon, y tiene los mismos permisos que el rol de usuario que hemos definido para este propósito en el servicio IAM de AWS. Este rol, de hecho, es un conjunto de permisos. El nombre de este rol para el RSC es “mmsclient”.

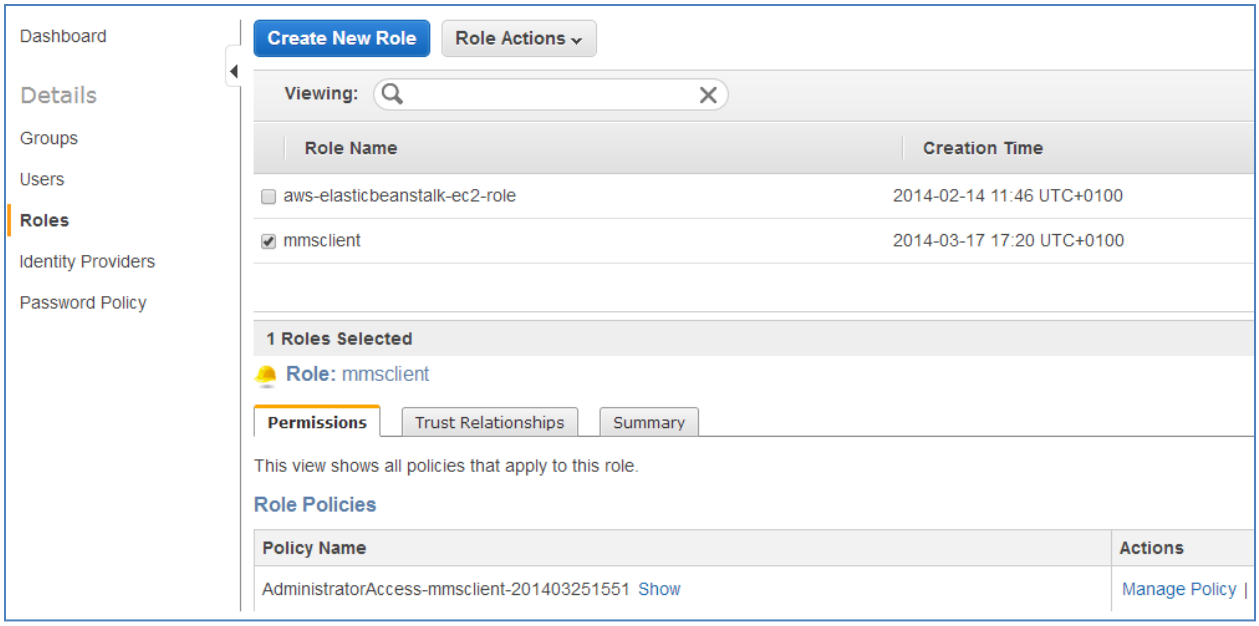

Figura 32: Rol mmsclient

Una vez el usuario ha conseguido asumir el rol mmsclient y el token de AWS, se procederá a la creación de la carpeta por defecto, si no está creada ya y el posterior envío del nombre de esta carpeta al cliente, o solamente el envío del nombre si ya está creada.

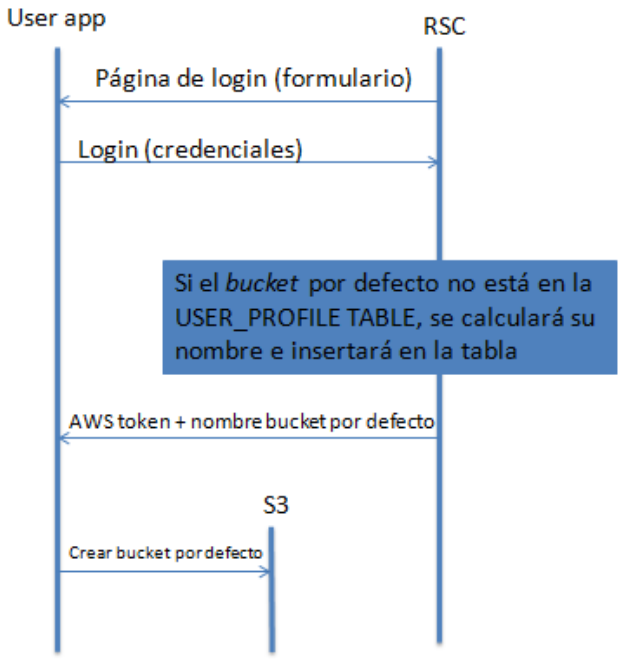

Figura 33: Flujo de acceso y autenticación

Autenticación delegada

El RSC permite el acceso mediante autenticación delegada mediante es estándar OpenID Connecto sobre el protocolo OAuth 2.0.

Para el caso del RSC el proveedor de identidad que se usa es Google, para lo que es necesario especificar el servicio en la consola de desarrolladores de Google como se ve en la Figura 34 y dar de alta a Google como proveedor de identidad en el servicio de Amazon IAM como se ve en la Figura 35.

Figura 34: Regsitro de la aplicación en la consola de Google

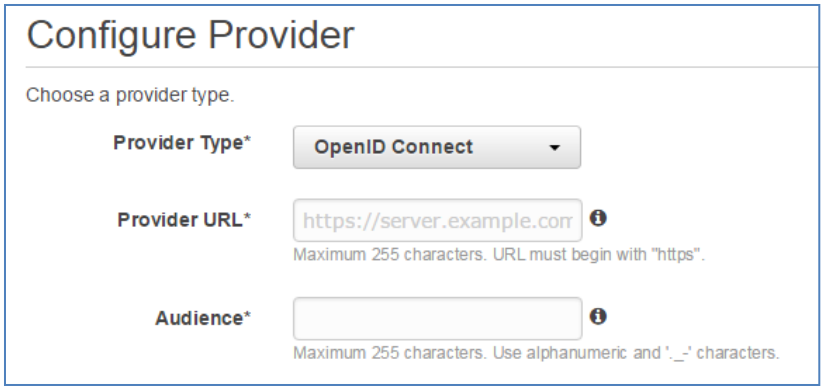

Figura 35: Dar de alta el proveedor de identidad

Una vez preparado el servicio para soportar la autenticación delegada el usuario puede loguearse con su cuenta de Google y este devolverá un token que se validará otra vez con el servidor de autorización de Google y entregará las credenciales de acceso a la aplicación como se ve en la Figura 36, con estas credenciales el RSC puede asumir el rol "mmsclient" y acceder a la funcionalidad del servicio.

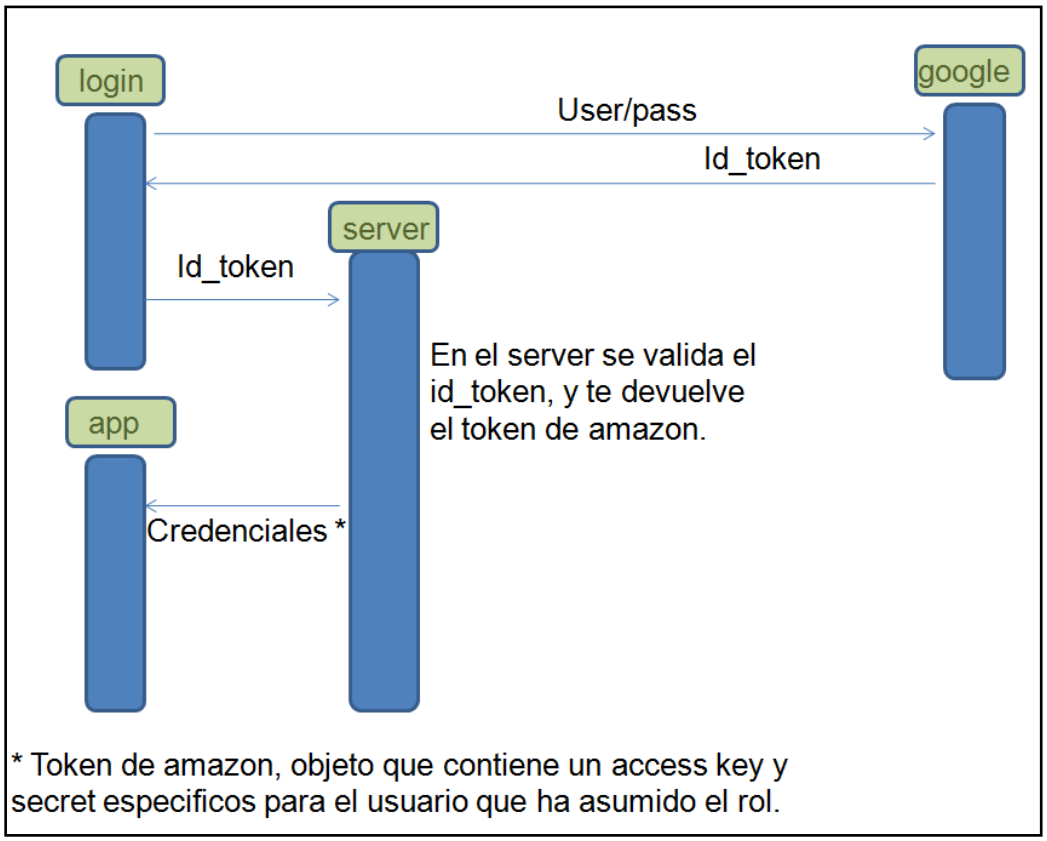

Figura 36: Autenticación delegada del RSC con Google como proveedor de identidad

3.2.1.2 Creación del bucket por defecto

La función que genera el nombre de la carpeta por defecto a partir de la identificación del usuario (user_id), no es una *función hash* secreta, ya que esto no es considerado un buen principio de diseño, siguiendo los principios de Kerchoff [61]:

- Si el sistema no es teóricamente irrompible, al menos debe serlo en la práctica.
- La efectividad del sistema no debe depender de que su diseño permanezca en secreto.
- La clave debe ser fácilmente memorizable de manera que no haya que recurrir a notas escritas.
- Los criptogramas deberán dar resultados alfanuméricos.
- El sistema debe ser operable por una única persona.
- El sistema debe ser fácil de utilizar.

Es recomendable asumir que el potencial hacker sabe todo sobre el método usado y que la única parte secreta es la clave criptográfica, ya que de otra manera un atacante podría deducir el método usado y atacar el sistema. Por esto se ha elegido la solución de generar el nombre de la carpeta por defecto, mediante un MAC (Código de Autenticación de Mensaje), que devuelve una cadena de tamaño limitado pero usando una clave, el secreto no es la *función hash*, el secreto es la clave.

Para construir el MAC se usa la función HMAC-SHA-256, basada en la *función hash* SHA, aplicándola sobre el usuario y una clave secreta obtendremos de forma unívoca una cadena que concatenada al nombre de usuario formará el nombre de su carpeta por defecto.

MAC= f (user_id, secret-key). Donde f es la función HMAC-SHA-256 y la secret key es un parámetro del servicio al que nadie puede acceder.

Una vez que se obtiene el nombre del bucket por defecto, este se guarda en la tabla USER_PROFILE_TABLE, y este valor se comprueba en la tabla en cada login, para prevenir que un cambio de clave secreta, haga que también se cree un bucket por defecto nuevo. Solo se calcula el nombre del bucket por defecto, si este no se encuentra en la tabla.

El bucket por defecto se crea directamente desde el cliente, comunicándose directamente con S3, el usuario no puede saber cuál es el nombre de su bucket por defecto.

Todos los buckets internos que el usuario cree sobre este bucket por defecto, se crean directamente por el cliente realizando funciones sobre S3.

3.2.1.3 Creación de carpetas

Dentro del bucket global del RSC, se encuentran los buckets por defecto de cada usuario, y dentro del bucket por defecto de cada usuario, este podrá crear nuevos buckets, pero nunca podrá crearlos fuera de este. En la Figura 37 podemos ver el árbol de buckets y sub buckets en S3 para el RSC.

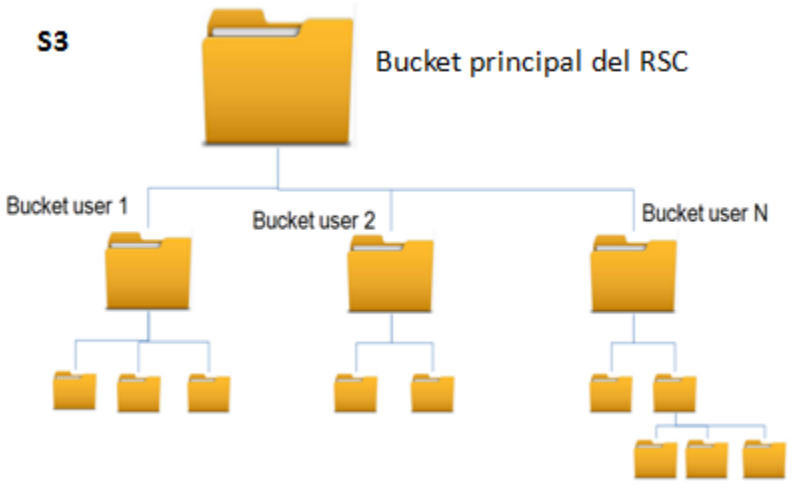

Figura 37: Jerarquía de buckets en S3

3.2.1.4 Subida de contenido multimedia

Un usuario puede subir ficheros invocando al servicio S3 directamente, mediante la función putObject() del API de S3, o borrarlos con la función deleteObject() del API de S3.

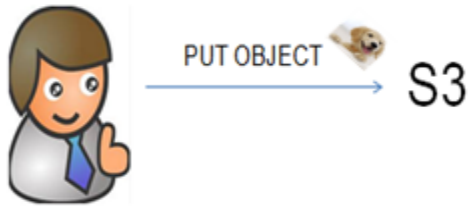

Figura 38: Interacción entre el cliente y S3 para subir objeto

Cada fichero subido tiene su propia URL en S3, que es de la forma:

```
https://S3-eu-west-1.Amazonaws.com/multimedia-sharing/<user_id_bucket>/objeto.txt
```

Un usuario solo puede subir ficheros dentro de su bucket por defecto, o de los buckets que haya creado dentro de este. La aplicación cliente siempre entra directamente dentro de tu bucket por defecto y no se puede salir de él, por lo que todas las operaciones son dentro de este bucket.

La subida de ficheros, o creación de buckets no afectan a la base de datos, ya que la tabla MMS_RESOURCE_TABLE solo contiene los recursos que han sido compartidos.

Los ficheros solos no se pueden compartir, lo que se comparte son las carpetas que los contienen, y las búsquedas responden con carpetas que se adaptan a los criterios de búsqueda, no los ficheros individuales.

3.2.1.4.1 Thumbnails

Cuando un usuario sube una imagen, se genera en el lado cliente a la vez un thumbnail, que se mostrara en este lado para no tener que cargar la imagen original siempre, que puede ser muy pesada, y así reducir la carga de imágenes.

La carga de una imagen generara la creación de un elemento canvas en el cliente, este canvas se rellenara con la imagen que acabamos de subir pero transformada para que la relación de aspecto del thumbnail sea el máximo de 128x128 o una cuarta parte de la relación de aspecto original, hacemos esto para no reducir demasiado la imagen y que se siga pudiendo ver en el lado cliente sin demasiada reducción de la calidad.

3.2.1.5 Creación de nuevos servicios

Un servicio a diferencia de otro tipo de recursos como una carpeta, consta de una descripción desde su creación (las carpetas solo tendrán descripción cuando sean compartidas), y además necesita más parámetros para una definición completa. Algunos de estos parámetros pueden ser funcionales (por ejemplo, una impresora puede tener una velocidad de impresión, gestión de colas de trabajo), y parámetros no funcionales (como el color de la impresora o su peso). Todos estos parámetros deben ser definidos en la creación del servicio y guardados en la tabla SERVICE_PROFILE_TABLE.

La definición de parámetros debe ser versátil porque cada servicio tiene los suyos propios, la única clasificación que se hace es la de funcionales y no funcionales, permitiendo cualquier número de cada tipo. En la figura 39 se ve como es el formulario de creación de un servicio, inicialmente permite dos parámetros de cada tipo, pudiendo aumentar este número, o reduciéndolo dejando el campo en blanco.

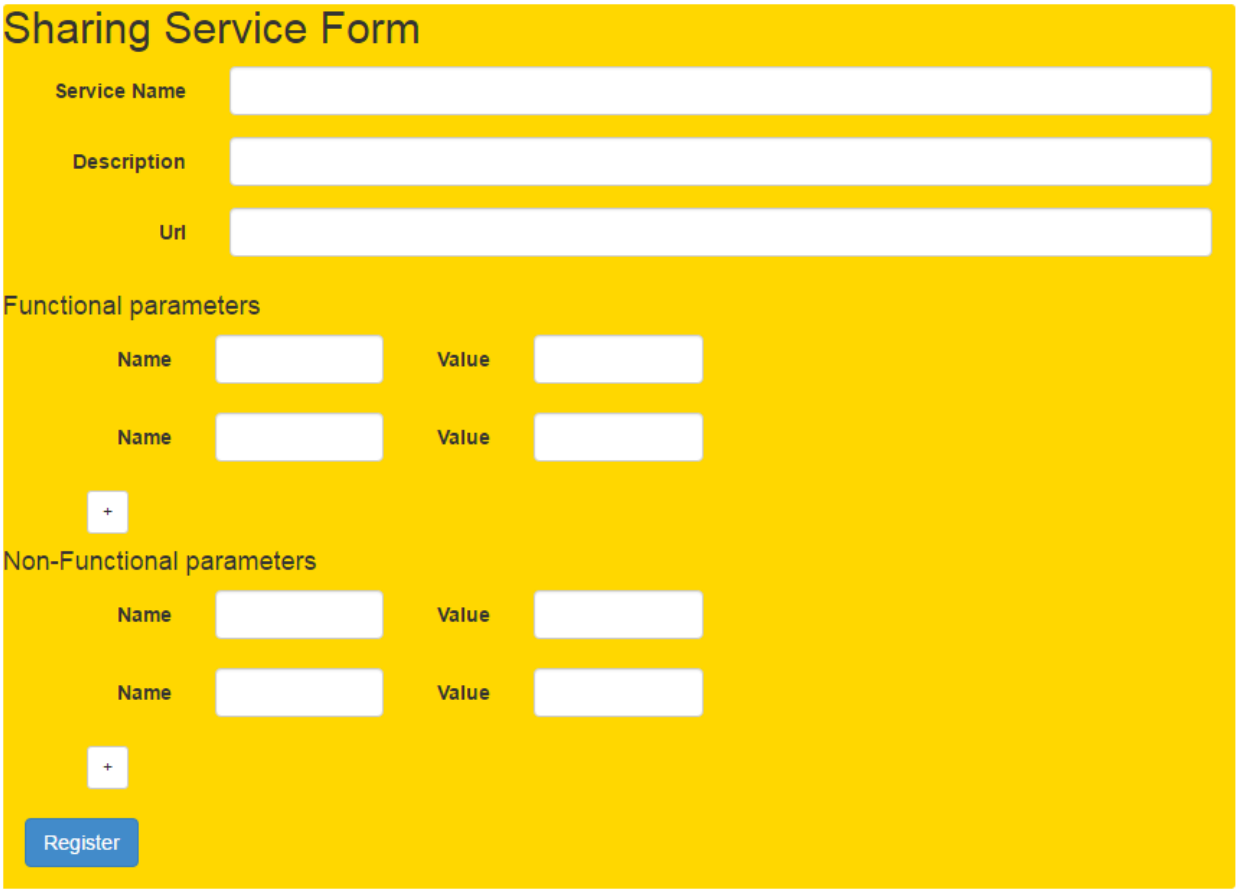

Figura 39: Formulario de creación de parámetros

Los servicios no se pueden almacenar en S3 como ocurre con las carpetas o ficheros, por eso, a pesar de no estar compartidos, se almacenan en la tabla SERVICE_PROFILE_TABLE como vemos en la Figura 40. La descripción se procesa para obtener la lista de palabras claves que se incluyen en el perfil de usuario.

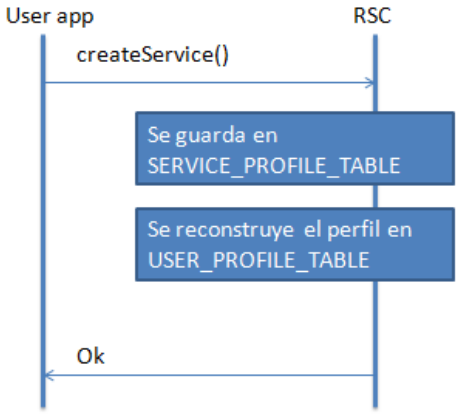

Figura 40: Flujo de creación de servicios

3.2.1.6 Compartición de recursos

Se pueden compartir cuatro tipos de recursos: Carpetas, servicios, links favoritos e intereses, el proceso de compartición consiste en elegir una serie de parámetros necesarios, como la url del recurso a compartir, el círculo en que se compartirá, y la descripción del recurso (para obtener las palabras clave que obedezcan al servicio de búsqueda) y compartir el recurso insertándolo en la tabla MMS_RESOURCE_SHARING_CENTER mediante la función shareResource() que el cliente invoca contra el RSC, como se ve en la Figura 41.

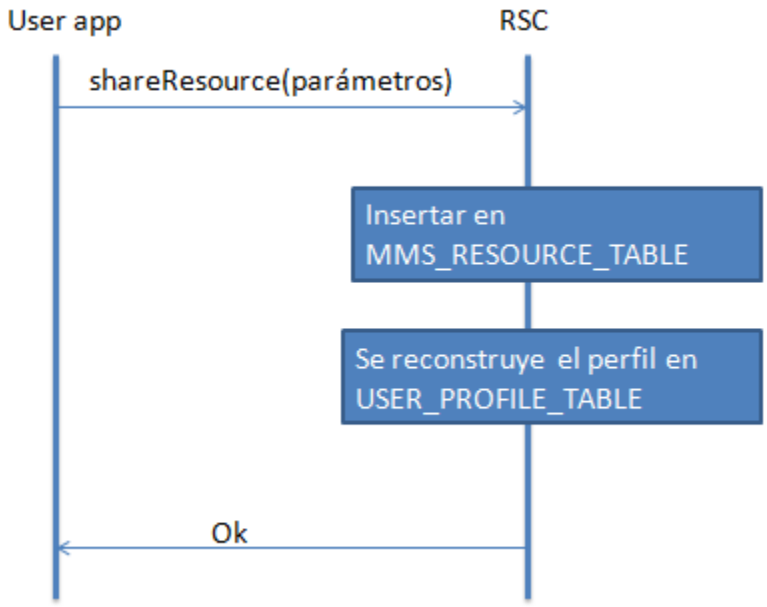

Figura 41: Compartición de recursos

La función shareResource() inserta o actualiza los atributos de un recurso compartido en la tabla MMS_RESOURCE_TABLE.

Cuando un usuario comparte un recurso es obligatorio proporcionar una lista de palabras clave asociadas al recurso, para insertarlas en la tabla MMS_RESOURCE_TABLE. Esto permite ejecutar búsquedas basadas en las claves y por lo tanto encontrar estos contenidos basándose en las claves proporcionadas.

La lista de claves se extrae del atributo descripción que el usuario proporciona durante el proceso de compartición usando un algoritmo lematizador, y tanto la descripción como la lista de claves se incluyen en la tabla MMS_RESOURCE_TABLE.

Algoritmo lematizador

Los metadatos de los objetos compartidos son generados aplicando un algoritmo lematizador sobre el campo descripción.

Sobre la descripción de los recursos se aplica un algoritmo de lematización de desmonte de sufijos, de la forma que se ve en la Figura 42.

```
//Paso 0: Pronombre adjunto. // Step 0: Attached pronoun
var pronoun_suf = new Array('me', 'se', 'sela', 'selo', 'selas', 'selos', 'la', 'le', 'lo', 'las', 'les', 'los', 'nos');
var pronoun_suf_pre1 = new Array('éndo', 'ándo', 'ár', 'ér', 'ír');
var pronoun_suf_pre2 = new Array('ando', 'iendo', 'ar', 'er', 'ir');
```

Figura 42: Reglas de desmontado de sufijos

Para llevar a cabo esto, hay que eliminar primero las s*top words* (palabras como: a, un, unas que no aportan información sobre la descripción) y luego aplicar las reglas de desmontado de sufijos sobre las palabras que quedan para así devolver una lista de las claves sobre las que se basarán las búsquedas (por ejemplo, si se busca cochecito, se buscará la raíz coch, para devolver también palabras relacionadas).

El algoritmo lematizador usado en este trabajo está basado en el algoritmo de Martin Porter, es la traducción a JavaScript de la implementación del algoritmo en php por Paolo Ragone [62].

Construcción y actualización del perfil de usuario.

La lista de palabras clave de los contenidos compartidos por un usuario particular forma parte de su perfil, por lo tanto un usuario está definido por la suma de las listas de palabras clave de sus objetos compartidos. Si hay más de una aparición de la misma palabra clave, se tiene en cuenta mediante un contador de cada palabra clave en el perfil del usuario que normalizamos respecto del total de las palabras y denominamos peso de la clave. En la Figura 43 vemos una representación gráfica de un perfil de usuario con sus claves y pesos asociados.

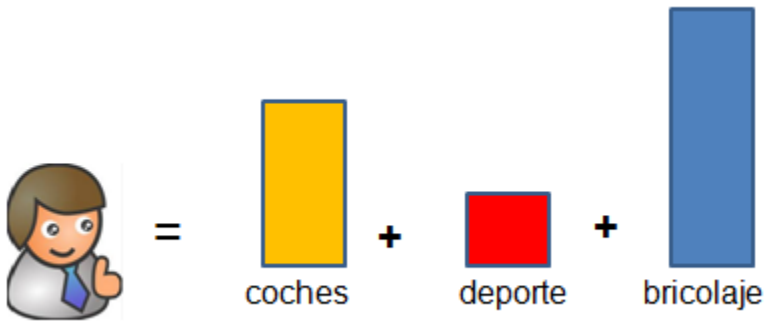

Figura 43: Representación del perfil de un usuario

El formato de la lista de palabras clave es JSON, a continuación vemos un ejemplo del atributo PROFILE de la tabla USER_PROFILE_TABLE:

[{"key":"MUSIC","weight":2},

{"key":"SHAKIRA","weight":1},

{"key":"LENNON","weight":1},

{"key":"COMPUTERS","weight":1}]

Cada vez que un usuario comparte un recurso, la función rebuildProfile() del RSC (que reconstruye el perfil del usuario en función de las nuevas palabras claves) es invocada, pero no desde el lado cliente, se hace automáticamente en el servidor cuando inserta un nuevo elemento en la tabla MMS_RESOURCE_TABLE.

Compartición de recursos: Carpetas

El método de compartición de ficheros se realiza compartiendo las carpetas que los contienen, no se pueden compartir los ficheros de forma independiente, ya que se comparte la url de S3 que da acceso al bucket particular representado por esa carpeta.

En la Figura 44 vemos el formulario de compartición de carpetas que incluye los campos url, communities (que es el círculo donde se compartirá) y descripción.

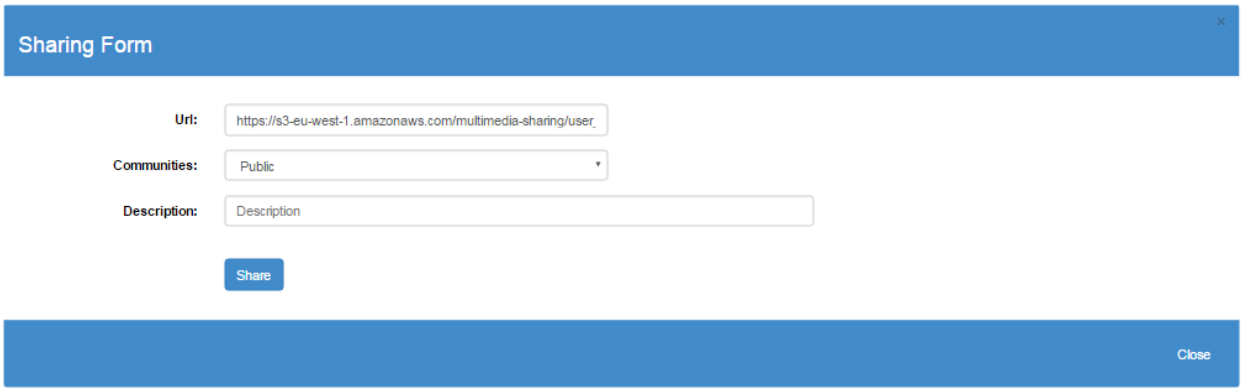

Figura 44: Ventana modal con formulario de compartición

Compartición de recursos: Links favoritos

Este tipo de contenido es útil para compartir videos de youtube, páginas web, aplicaciones web y otro contenido público accesible desde una url sin necesidad de subir ningún objeto a S3.

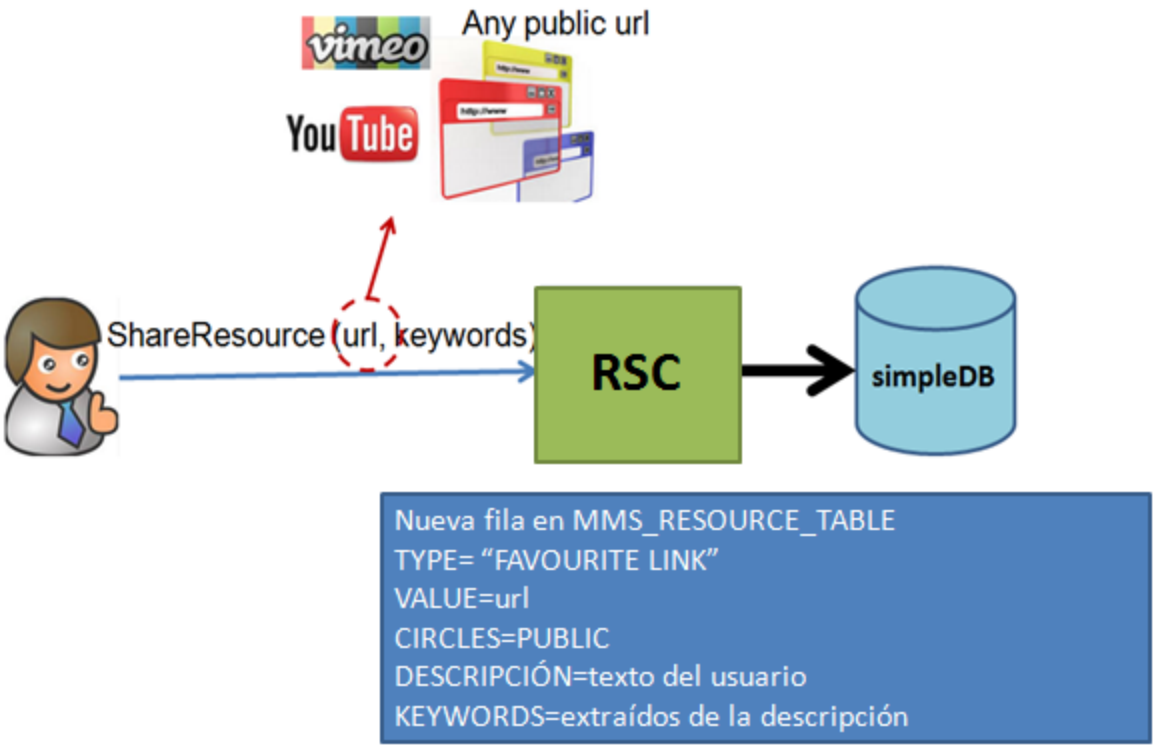

Figura 45: Compartición de links favoritos

El círculo en que se comparten los links por defectos es el círculo PUBLIC por defecto, ya que el usuario no suele ser el dueño de este contenido y no hace falta restringirlo a algún círculo.

Después de la inserción de la fila en la tabla MMS_RESOURCE_TABLE, el perfil del usuario se reconstruye de forma automática desde el lado servidor, sin que haga falta que el cliente invoque a la función rebuildProfile().

Compartición de recursos: Servicios

Un servicio, además de una descripción y lista de palabras clave a partir de esta necesita más parámetros para estar completamente definido, estos parámetros son los denominados parámetros funcionales y no funcionales que se describen en el proceso de creación del servicio.

Por lo tanto solo la descripción, el campo communities (el círculo donde se compartirá) y la url del servicio, están disponibles en el formulario de compartición, ya que el resto de parámetros para definir el servicio están definidos desde la creación. El campo descripción contendrá la descripción que se usó para crear el servicio, y permite actualizarla. La Figura 46 muestra como es este formulario de compartición.

Figura 46: Formulario de compartición de servicios

El proceso de compartición es el mismo que ya se ha descrito, desde el cliente se invoca shareResource() que añade una nueva fila a MMS_RESOURCE_TABLE y se reconstruye el perfil del usuario.

Los cambios en la descripción del servicio que se hagan, solo afectan a las búsquedas y al perfil del usuario al compartir el servicio, porque se insertan en MMS_RESOURCE_TABLE, de lo contrario (al crear o actualizar el servicio), solo están en SERVICE_PROFILE_TABLE.

Compartición de recursos: Intereses

El proceso de compartir intereses es el mismo que ya se ha visto para el resto de recursos, pero en este caso el campo VALUE que insertará en la tabla MMS_RESOURCE_TABLE será INTEREST, ya que el valor de compartir un interés está en la descripción del mismo. Como vemos en la Figura 47 el formulario de compartición solo incluye el campo descripción, los intereses son siempre públicos y sirven para actualizar tu perfil.

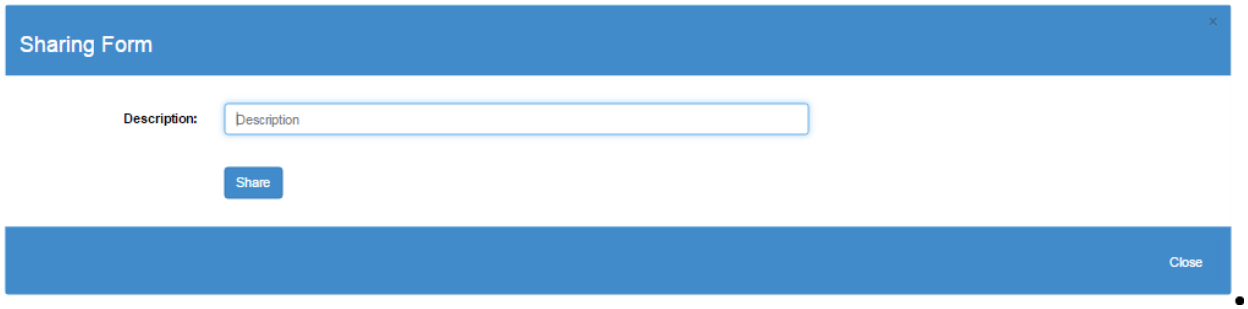

Figura 47: Formulario de compartición de intereses

La invocación de shareResource() con TYPE = "INTEREST" se produce desde el menú del perfil de usuario en la parte cliente de la aplicación, como se ve en la Figura 48.

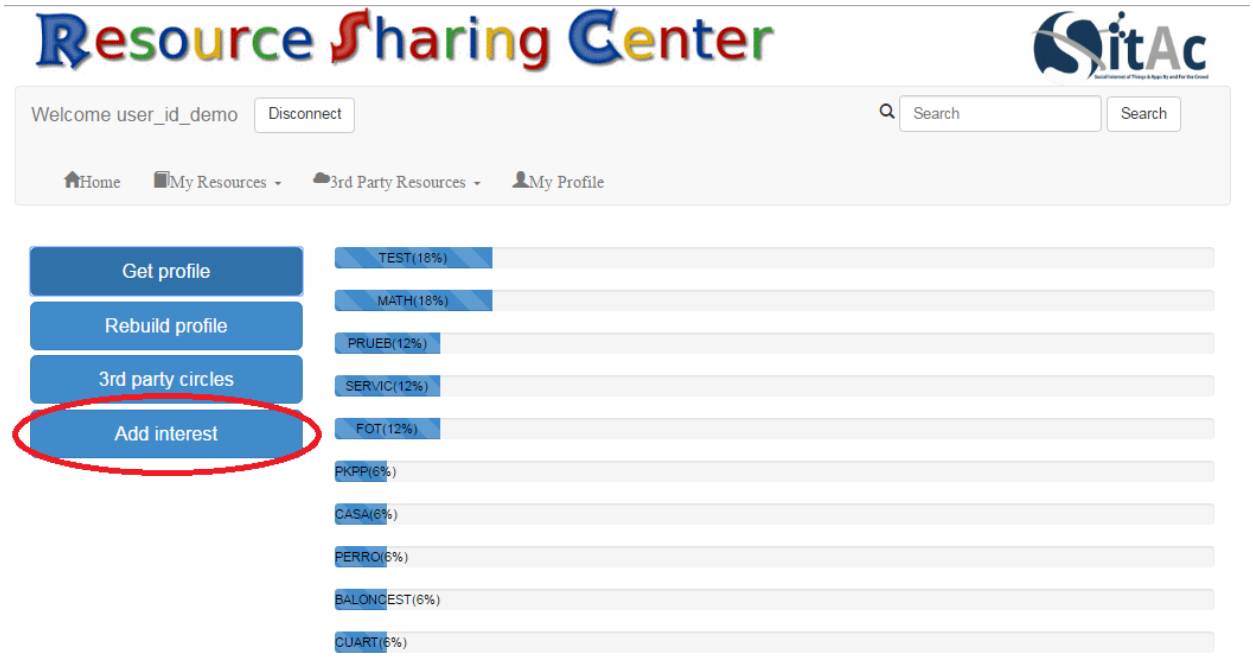

Figura 48: Invocación de añadir interés desde el lado cliente.

3.2.1.7 Consultar el inventario

El usuario tendrá dos inventarios dentro de la aplicación, el inventario propio y el de terceros, con los recursos que hayan sido compartidos con él.

La consulta del inventario propio se basa en invocaciones a funciones de S3 para ver el inventario de recursos lógicos (carpetas y ficheros), llamadas a la función getServices() para obtener los servicios creados por el usuario, y a getSharedResources() para ver los links favoritos que has compartido.

La consulta del inventario de terceros se basa por completo en invocaciones al back-end de la aplicación, ya que todos los recursos de terceros que han compartido contigo, se encuentran en la tabla MMS_RESOURCE_TABLE con el campo OWNED=”NO”, por lo que hay que realizar llamadas consecutivas a getSharedResources().

3.2.1.8 Servicio de búsqueda

Las búsquedas son posibles gracias al campo KEYWORDS de la MMS_RESOURCE_TABLE, ya que en él se encuentra el resultado de la lematizacion de la descripción de los recursos compartidos.

El servicio de búsqueda se puede realizar de dos maneras, haciendo full scan de la tabla, lo que es menos eficiente y construyendo una tabla con el índice inverso y recorrer esa tabla.

- *Full scan*: La búsqueda que se lanza contra la base de datos se retornan los ítems que contienen todas las claves que buscamos. Esto representa una limitación pero simplifica el código a una sola consulta que es muy rápida, a pesar de que no aprovechamos el índice automático de SimpleDB, ya que la búsqueda se hace sobre la lista de claves, y no sobre un elemento único que sí se puede indexar.

- Con índice inverso: Hacemos la búsqueda sobre la tabla MMS_RESOURCE_INVERSE, en la que los elementos están ordenados por una de las palabras clave y no por el nombre del objeto, esto es, los elementos pueden aparecer repetidos, pero la búsqueda es mucho más rápida ya que podemos aprovechar el índice automático de SimpleDB, y luego ordenar las respuestas en función del valor MMS_RANK del ítem con más claves en la respuesta. Con esta opción la tabla MMS_RESOURCE_INVERSE debería actualizarse cada cierto tiempo.

3.2.1.9 Acceso a contenidos de terceros

El acceso a contenido de terceros sobre los resultados de una búsqueda comprende dos escenarios: el usuario con permiso al recurso, y el usuario que no tiene acceso al mismo, esto es, que no pertenece a los círculos en que se ha compartido el recurso.

El contenido de terceros que se muestra al realizar las búsquedas está basado en el nombre del objeto en la tabla MMS_RESOURCE_TABLE, que consiste en la concatenación del USER_ID del dueño del recurso con el campo VALUE, del que solo se muestra el nombre del servicio al final de la URL que es el VALUE, para no mostrar la url específica de S3 que contiene información que el usuario no debe conocer, como el nombre del bucket por defecto.

Hay un caso especial de acceso a contenidos de terceros, que ocurre cuando el contenido esta compartido en el círculo Market, este caso se contempla en el siguiente apartado.

Acceso a contenidos de terceros con permiso

Después de que un usuario pida acceso a un recurso en particular mediante la función pickup() sobre el contenido devuelto tras una llamada a la función search(), el servidor comprueba si el usuario pertenece a algún círculo en que el recurso esté compartido, en caso de que sea así el recurso se incluirá en la tabla MMS_RESOURCE_TABLE con el valor OWNED = “NO” y USER_ID= “User id del usuario que ha realizado el pickup() sobre el recurso”. De forma que este recurso se incorpore al inventario de terceros del usuario que hace el pickup().

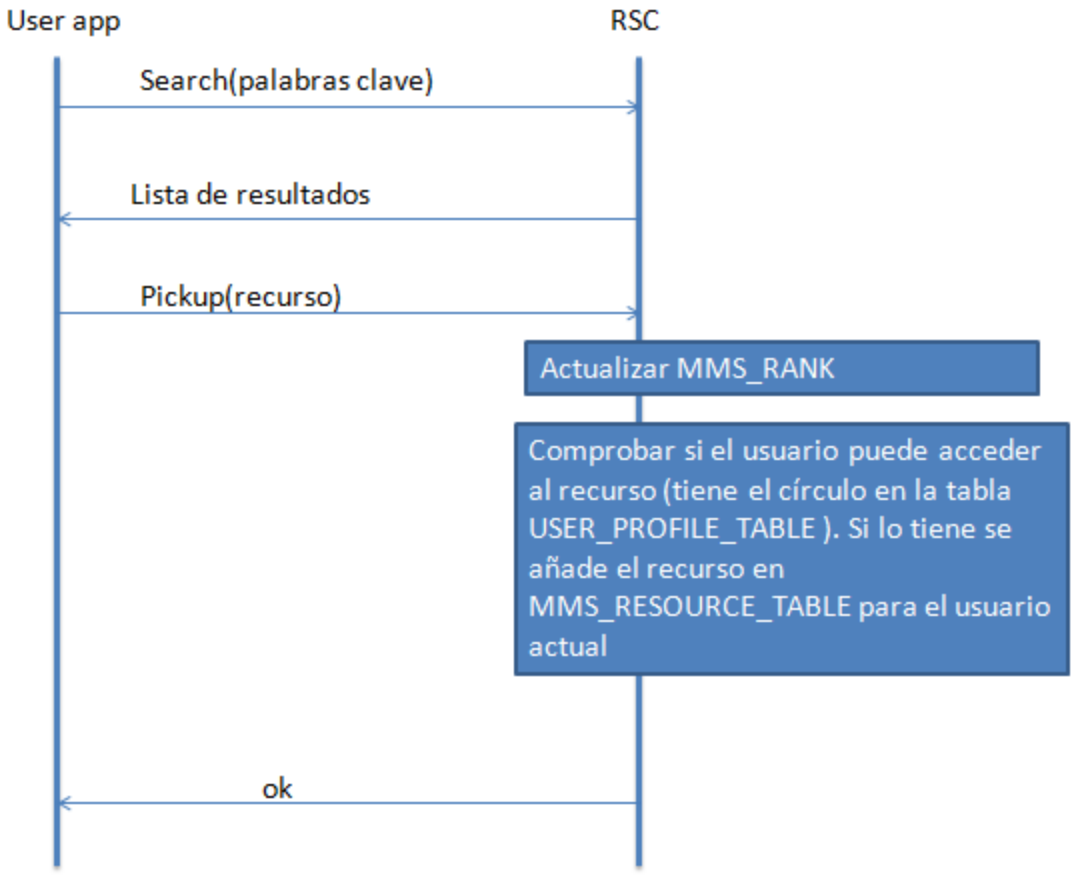

Figura 49: Acceso a contenido de terceros con permiso

Acceso a contenidos de terceros sin permiso

El Resource Sharing Center comprueba si el usuario tiene o no acceso al recurso, basado en los círculos en los que esta compartido, en el caso en que el usuario no tenga acceso, el servicio le permitirá solicitar acceso a alguno de los círculos en los que esta compartido.

La aplicación en el lado cliente invoca en este caso la función getCirclesSharedResource() que devuelve la lista de círculos en que esta compartido un recurso, y te ofrece la opción de pedir permiso al dueño del circulo para acceder al mismo mediante la función requestMembership(), que desde el cliente envía un mail al dueño con un link que el dueño puede pulsar para permitirle acceso al círculo requerido, como se ve en la Figura 50.

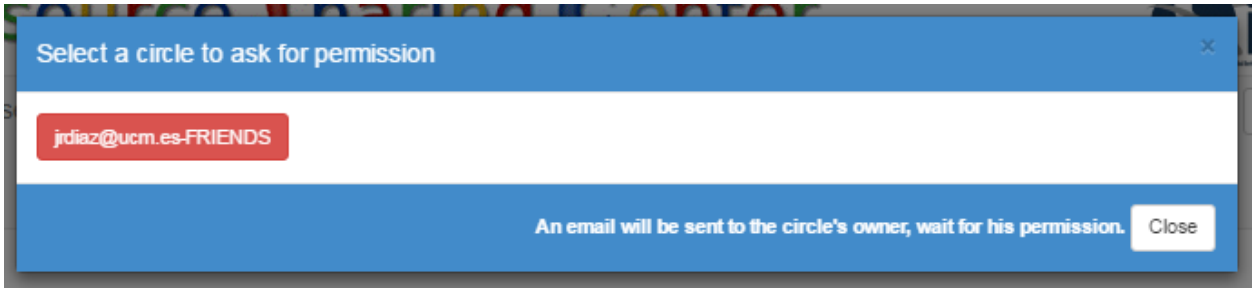

Figura 50: Ventana modal con los círculos en que esta compartido un recurso.

En la Figura 51 se puede ver el proceso completo.

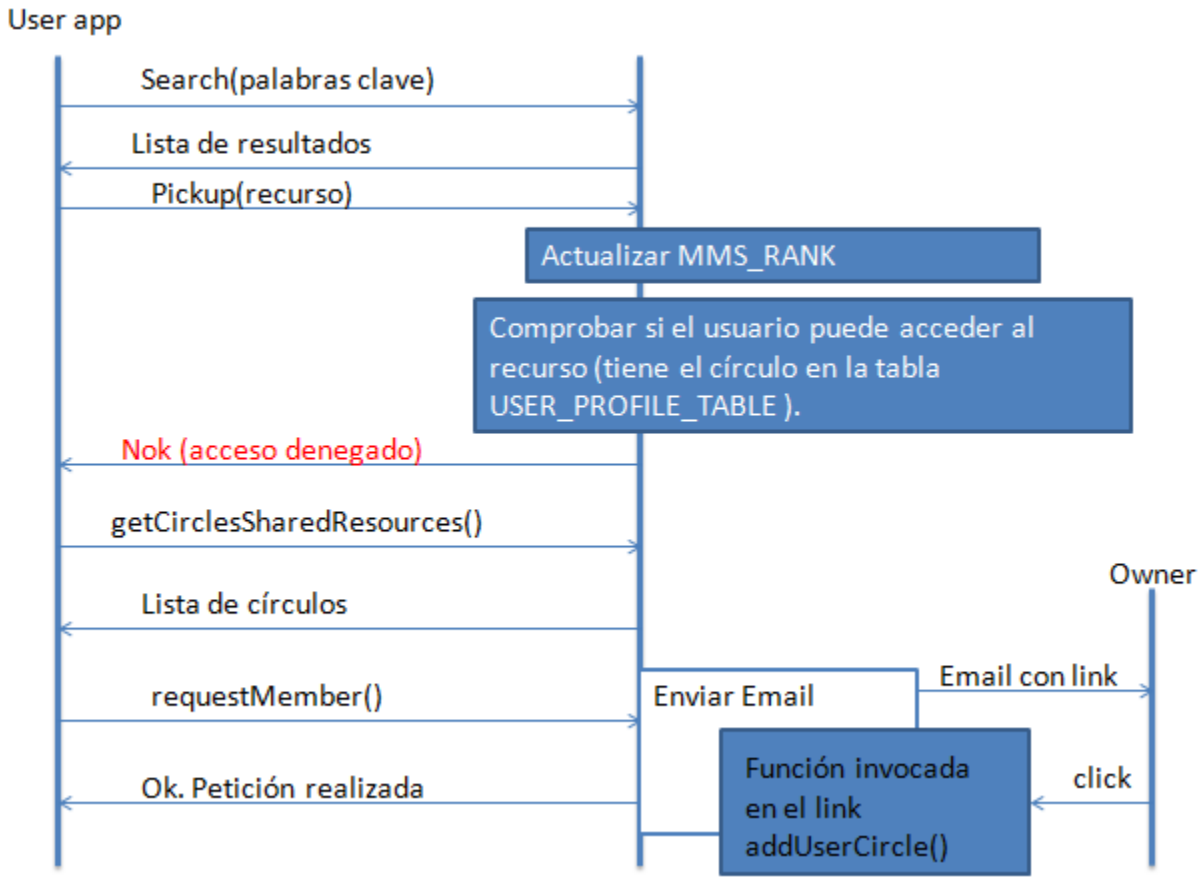

Figura 51: Acceso a un recurso sin permiso

Descarga de contenidos una vez accedes a ellos

Una vez accedes a un contenido de terceros, si estos son ficheros o contenidos multimedia puedes descargarlos directamente invocando la API de S3, y si es un link favorito puedes acceder directamente a la url.

3.2.1.10 Uso del marketplace

El Resource Sharing Center proporciona la posibilidad de vender los recursos que compartes, la forma de hacer esto es compartiéndolo en un círculo llamado "MARKET".

Cuando un usuario comparte una carpeta o servicio en este círculo, es obligatorio añadir el atributo precio, como se ve en la Figura 52.

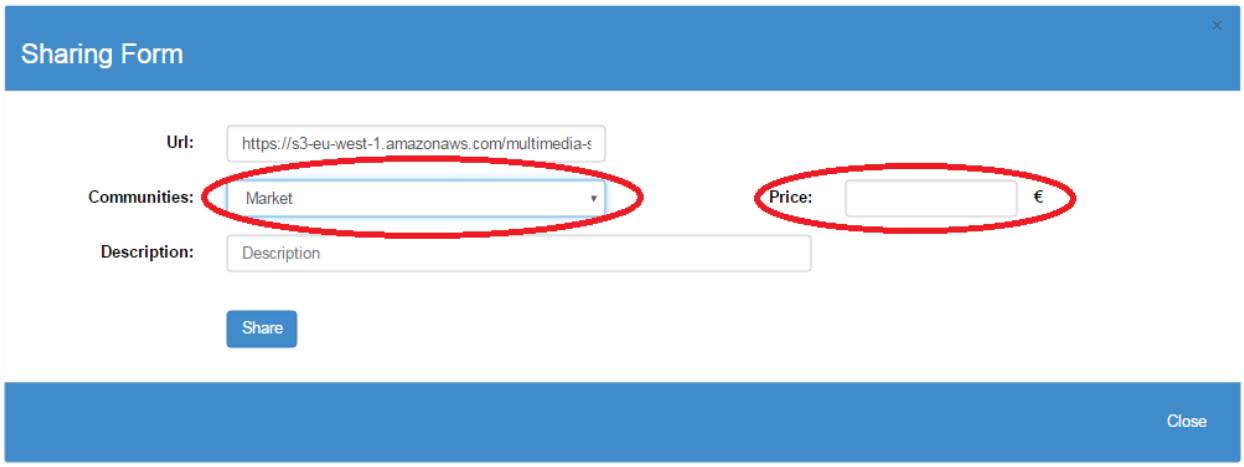

Figura 52: Formulario de compartición en círculo market

Para la invocación de shareService() el nombre del circulo Market es reemplazado por el mismo concatenado con una cadena aleatoria, para que cada círculo Market sea único.

Cuando un usuario hace pickup() sobre un recurso compartido en el círculo Market, se le invita a pagar el precio mediante la función pay(), que incluye al usuario en ese círculo Market específico y le añade el recurso a su inventario de terceros, llama internamente a addUserCircle() y a pickup(). En la Figura 53 se muestra este proceso.

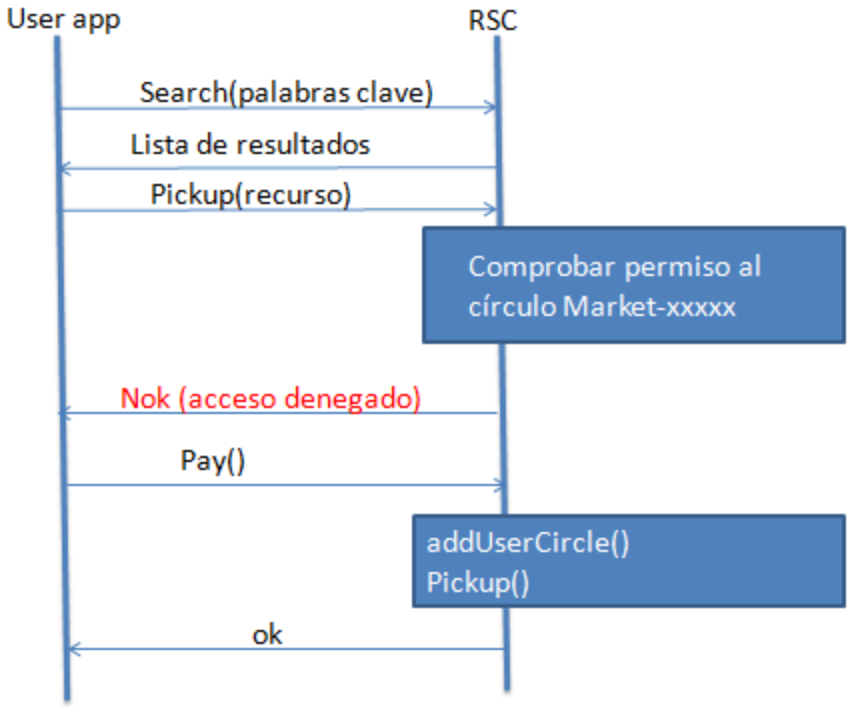

Figura 53: Acceso a recurso de pago

3.2.2 API

En esta sección se describen las funciones implementadas en los distintos módulos del servidor.

3.2.2.1 Módulo sharing (Node.js)

Las funciones definidas en el módulo sharing son las siguientes:

- searchResources(param user_id,param keywords)
  Admite búsquedas de varias keywords, especificadas como una lista.
  Retorna una lista JSON de items que cumplen la condición.

- getSharedResources(param user_id, [param type] , [param value], [param owned])
  Esta función retorna los elementos de la tabla MMS_RESOURCE_TABLE que pertenecen al usuario y que poseen el TYPE especificado y/o el VALUE especificado.

- shareResource(param user_id, param type, param value, param circles, param description, param keywords, param [price], param [service_name])

Sirve para insertar un nuevo recurso, además, esta función de forma automática invoca a rebuild profile, de modo que ya se queda actualizado el perfil.

- deleteResource(param user_id, param value)
  Borra el item cuyo itemName es la concatenacion de userid+value
  además esta función de forma automática invoca a rebuild profile, de modo que ya se queda actualizado si el usuario que borra es el owner, además se borran todas las tuplas donde aparezca como 3rd party folder, es decir, donde aparezca como owner con user_id <> owner

- pickupResource(param user_id, param item_name, param email)
  Inserta un registro en la tabla MMS_RESOURCE y le pone owned=NO

- getProfile (param user_id, [top_n])
  Retorna el perfil del usuario, concretamente los campos PROFILE, OTHER_CIRCLES y OWN_CIRCLES de la tabla USER_PROFILE_TABLE

  Si el parámetro top_n llega, se tiene en cuenta. Por defecto se asigna un 10, de modo que se contesta con las 10 keywords de mayor peso

- rebuildProfile(param user_id)
  Reconstruye el perfil del usuario. No retorna el perfil, solo una indicación de OK/NOK, ya que el perfil se deja en background construyéndose y se retorna el control a la aplicación.

- searchSharedResourcesInv2: Es la búsqueda basada en la tabla de índice inverso, es mucho más rápida.
  .

3.2.2.2 Módulo hello (Python)

Las funciones definidas en el módulo hello son las siguientes:

- getMac(user, clave)
  En función de los dos parámetros de entrada codifica el nombre de la que será la carpeta por defecto.

- getAWSToken(token)
  Devuelve el token temporal de aws mediante la función sts_connection.assume_role()

- getAWSToken_google(token_google)
  Igual que la anterior, pero cuando te logueas con google, la función con la que pides el token temporal es: sts_connection.assume_role_with_web_identity()

- hello()
  Esta es la función que hace todo el trabajo, una vez te has logueado, construye o busca tu bucket por defecto, y devuelve la página del servicio.

- hello_google()
  Función análoga a la anterior, pero con los parámetros de google, para este tipo de usuarios la codificación del nombre del bucket por defecto contiene un código distinto al de los usuarios que hacen login automático.

- requestMembership()
  Esta función es la que se encarga de enviar un mail al usuario dueño del círculo, al que se quiere entrar, el mail contiene un botón para permitir el acceso al círculo requerido.

3.2.2.3 Módulo batch

- buildinvidex(): Construcción del índice inverso.

3.3 Diseño cliente (GUI)

En este apartado se describirán las funciones y la GUI que se han usado para modelar el servicio cliente de esta aplicación.

3.3.1 Interacción con el back-end

Los scripts aquí descritos son llamados desde la página principal para comunicarse con el back-end, y en función de la respuesta recibida, contienen las funciones necesarias para generar dinámicamente la forma visual en el cliente de las respuestas JSON.

- llamada_search: Llama a la función search() descrita en el back-end y con la respuesta construye dinámicamente una tabla con los resultados.
- ajaxLlamadaGetProfile: Llama a la función getProfile() y con la respuesta construye de forma gráfica el perfil del usuario.
- llamadaGetSharedResourcesColor.js: Llama a la función getSharedResources() y en función de la respuesta pinta el icono de las carpetas en la tabla de tus ficheros de verde si están compartidas.

- sharingfunctions.js: Llama a la función share y controla el formulario de compartición que aparecerá en función de lo que se comparta. Si el círculo es MARKET, también genera la cadena aleatoria que se usará para identificarlo.
- llamada_PickUp.js: Llama a la función pickup().
- llamada_3rdParty.js; Llama a getSharedResources() y construye una tabla análoga a la de recursos lógicos en función de la respuesta.
- llamadaRebuildProfile.js: Llama a rebuildProfile()
- llamadaDeleteResource.js: Llama a la función deleteResource() para dejar de compartir un fichero.
- llamadaGetCirclesSharedResource.js: Llama a getCirclesSharedResource() para obtener los círculos en los que esta compartido un recurso, genera dinámicamente la tabla con los círculos y ofrece la opción de solicitar acceso vía email.
- llamadaGetSharedResourcesBorrar.js: Llama a getSharedResources() para comprobar si el recurso esta compartido antes de borrarlo.
- llamadaCreateService.js: Llama a createService() para crear un servicio.
- llamadaGetServices.js: Llama a getServices(), sobre la respuesta genera dinámicamente la página que te muestra los servicios.
- llamadaGetSharedResourcesFL.js: Llama a getSharedResources() para generar sobre la respuesta y de forma dinámica la tabla con los links favoritos del usuario.
- llamadaPay.js: Llama a la función pay() para pagar por un recurso de pago.

3.3.2 Descripción de la GUI

En este apartado se describirá la interfaz de usuario de la aplicación, está realizada en HTML5 con JavaScript para modelar la generación dinámica de los distintos aspectos de la página, y basada en la rejilla de filas y columnas de Twitter Bottstrap, para dar una visibilidad uniforme a la página, donde las distintas secciones se basan en un mismo diseño, manteniendo una barra de navegación en la parte superior de la página, mientras que el resto está dividido en dos secciones, una a la izquierda más estrecha, en la que se verán la posibles funcionalidades de cada sección, y otra más ancha a la derecha, en la que se verán los resultados de estas funcionalidades.

3.3.2.1 Página de login

Antes de acceder a la aplicación se muestra una página de login realizada con Twitter Bootstrap, con un formulario que se adapta al tamaño de la pantalla y con tres opciones para loguearte, el acceso directo, y el acceso con autenticación delegada, ya sea por parte de google, o con otra plataforma cuyo desarrollo no forma parte de este trabajo. Vemos la página de login en la Figura 54.

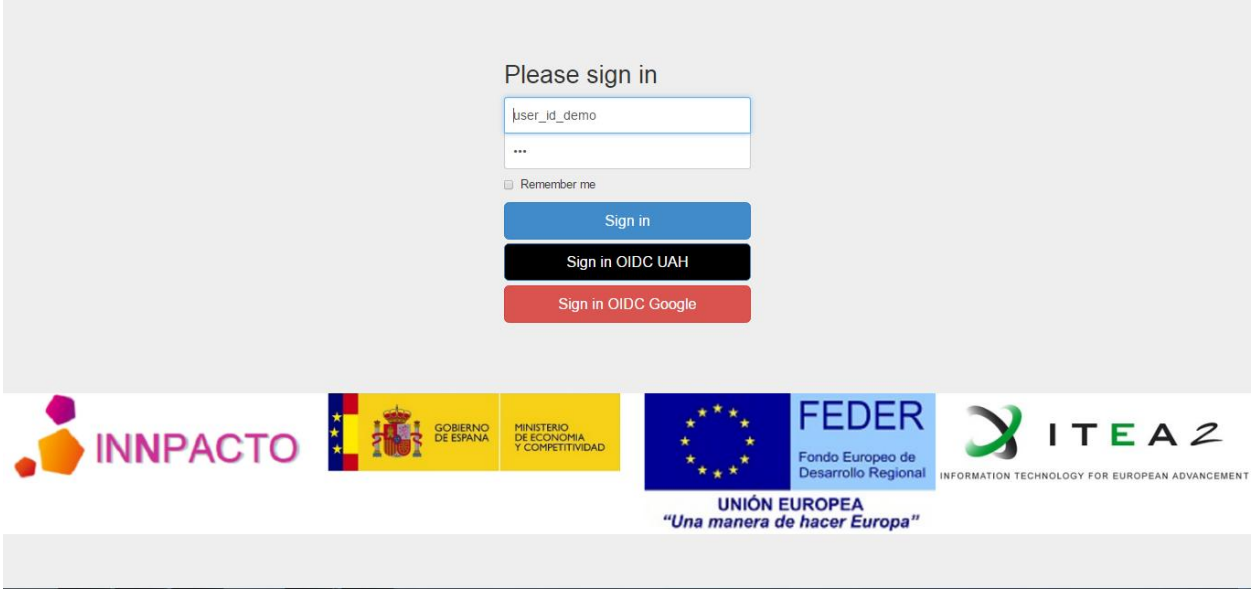

Figura 54: Página de login

3.3.2.2 Landing page

Una vez te has logueado, al cargar la página se genera dinámicamente una tabla con una recomendación basada en el clustering llevado a cabo por el módulo Analytics. La tabla se genera dinámicamente usando JavaScript sobre el DOM y con los estilos de Twitter Bootstrap. Esta página se ve en la Figura 55.

Figura 55: Página de aterrizaje

3.3.2.3 Barra de navegación y menús

La barra de navegación de la parte superior de la página es un elemento navbar con los estilos de Bootstrap se compone de tres partes, que se muestran en la figura 56:

- Welcome: Welcome + user_id, el user id se obtiene en el login y se lleva esa variable a la página.
- Menus: Hay cuatro pestañas: home, que te lleva a la página de inicio; My resources, donde puedes ver tus inventarios, de ficheros, links favoritos y servicios; 3rd Party Resources, que te lleva a los ficheros y servicios de terceros que han compartido contigo; My Profile, que te lleva a la página de gestión de tu perfil.
- Búsqueda: Un formulario con los estilos de bootstrap que está dentro de la navbar.

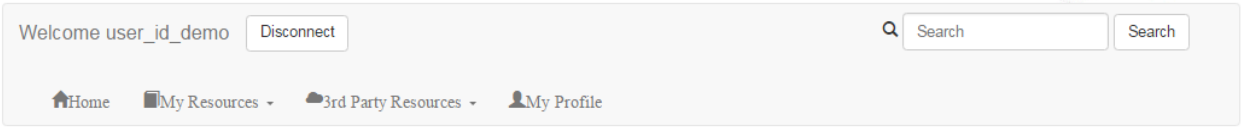

Figura 56: Barra de navegación y menús

3.3.2.4 My Resources: Files

Genera dinámicamente una tabla con tus ficheros, y un carrusel que muestra los elementos del fichero que estás viendo actualmente (Figura 57).

El código para generar la tabla y el carrusel están en los scripts: files_table.js y carousel2.js

A parte en a esta tabla se le añade un menú contextual personalizado que te da opciones específicas para cada elemento de la tabla sobre el que hagas click derecho. El código que controla este menú contextual alternativo esta en el script contextmenu.js y está basado en el código de Kyle Mitofsky [63].

Esta parte de la página permite subir ficheros y crear carpetas y que el resultado se compruebe al momento, sin tener que recargar la página.

Cuando se sube una foto se genera también el *thumbnail* de la misma para mostrarlo en el carrusel.

Figura 57: My Resources: Files

Las carpetas con un icono azul no están compartidas, mientras que las carpetas con un icono verde sí que lo están.

Cuando pulsamos el botón share del menú contextual se nos abrirá la ventana modal de compartición (Figura 58), en la que podremos elegir en que círculos compartir cada recurso, y si elegimos market aparecerá una opción para poner el precio.

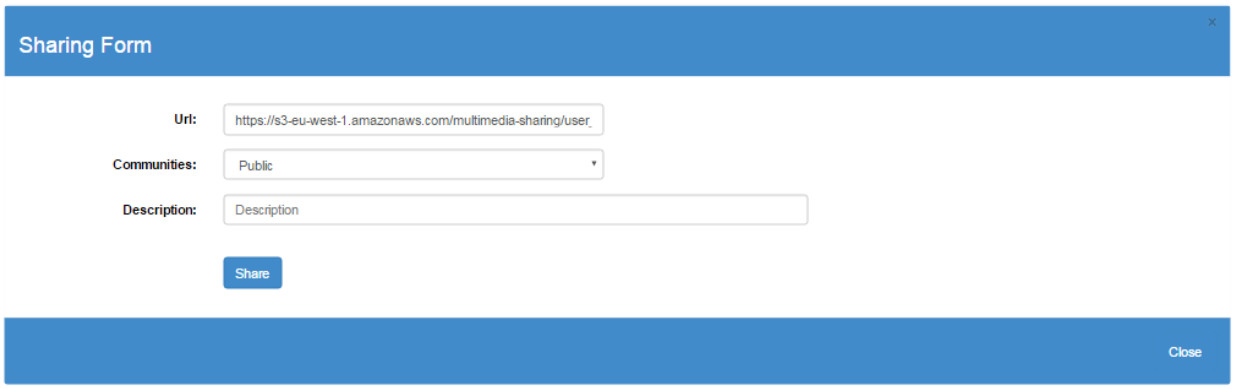

Figura 58: Ventana modal con formulario de compartición

3.3.2.5 My Resources: Favourite Links

En esta sección de la página hay un botón para añadir el link favorito, que abrirá una ventana modal para incluirlo, y automáticamente la pagina lo incluirá en la tabla de la derecha sin que haga falta recargarla (Figura 59), en esta tabla se mostraran como botones que al pulsarlos te abren en otra pestaña el link favorito.

Figura 59: Links Favoritos

El formulario de compartición de la Figura 60 es una ventana modal de compartición solo aparecen los campos url y descripción, porque el campo comunities es público por defecto, aparece oculto.

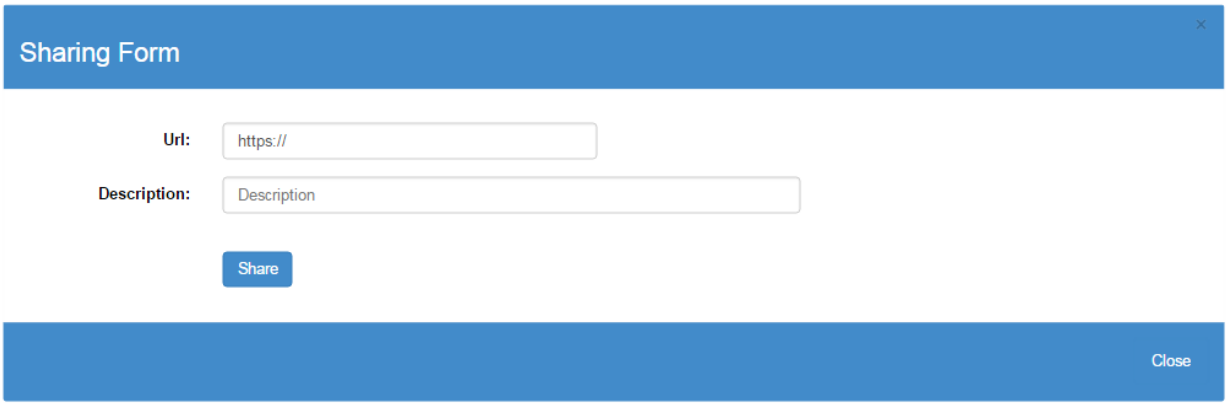

Figura 60: Formulario de compartición de Link Favorito

3.3.2.6 My Resources: Services

En esta vista tendremos a la izquierda un botón para crear un nuevo servicio, y debajo botones para los servicios ya creados, cada vez que pulsemos un botón en el formulario vacío de la derecha se nos mostrará la descripción del servicio.

El formulario de la derecha, puede estar vacío, lo que nos indicara que podemos crear un nuevo servicio, y solo ofrecerá la opción de registrarlo, en un botón al final del formulario, una vez registrado, nos mostrara las opciones de ir a la url del servicio, compartirlo, dejar de compartir actualizarlo o borrarlo como se puede ver en la Figura 61.

Figura 61: Formulario de alta de servicio

3.3.2.7 3rd Party Resources: Files

En esta parte de la aplicación se puede ver el inventario de ficheros que han compartido con nosotros. En la parte de la izquierda hay una tabla con las carpetas de distintos usuarios que tenemos, solo podemos acceder a ellas o borrarlas. Una vez se accede a ellas se muestran como en el inventario de ficheros propios, pero sin el menú contextual personalizado, ya que esos objetos no son realmente tuyos, solo puedes descargarlos al pinchar sobre ellos, como se ve en las Figuras 62 y 63.

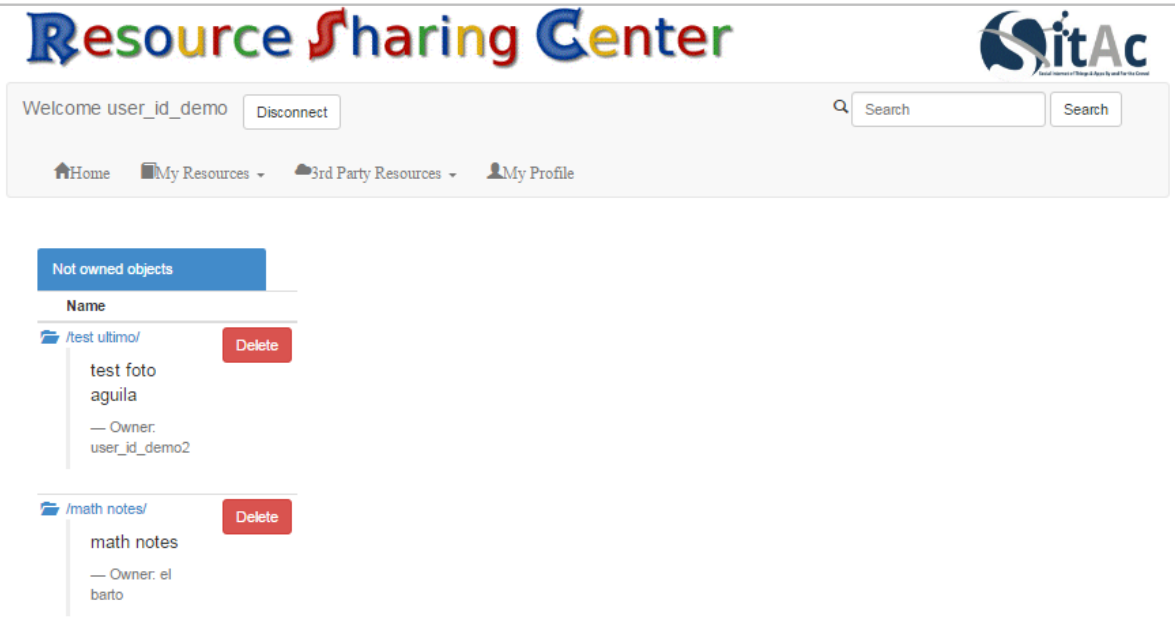

Figura 62: Carpetas de terceros I

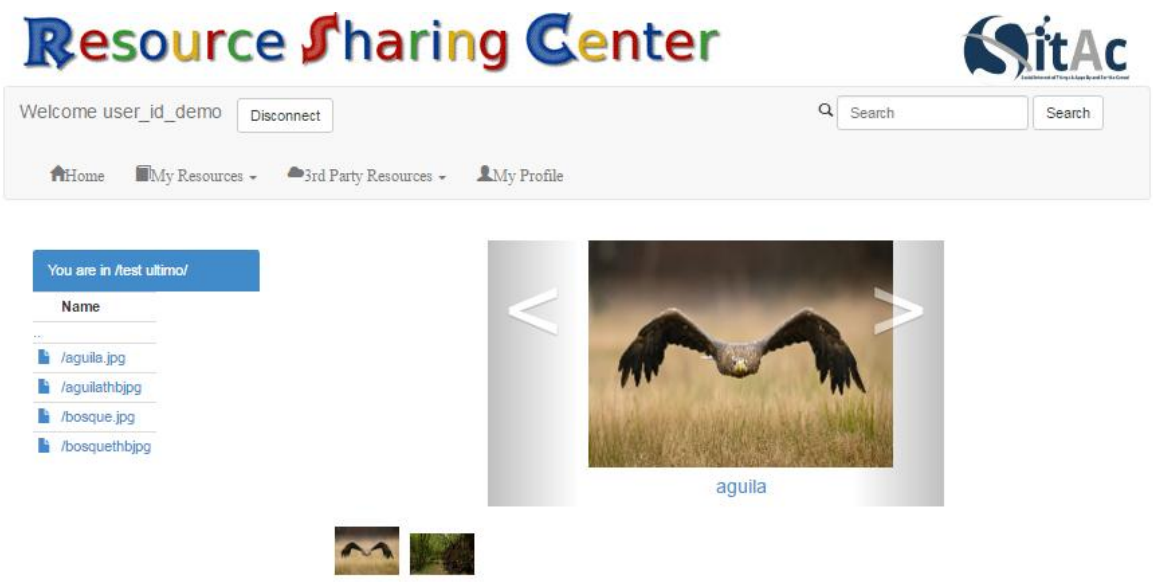

Figura 63: Carpetas de terceros II

3.3.2.8 3rd Party Resources: Services

La vista de los servicios de terceros se muestra en la Figura 64, es similar a la vista de tus propios servicios, salvo que desde aquí no puedes crear servicios nuevos, ni puedes compartir los

que ya te han compartido, ya que no son tuyos. Solo puedes acceder a ellos y borrarlos, esto es, eliminarlos de tu inventario de terceros.

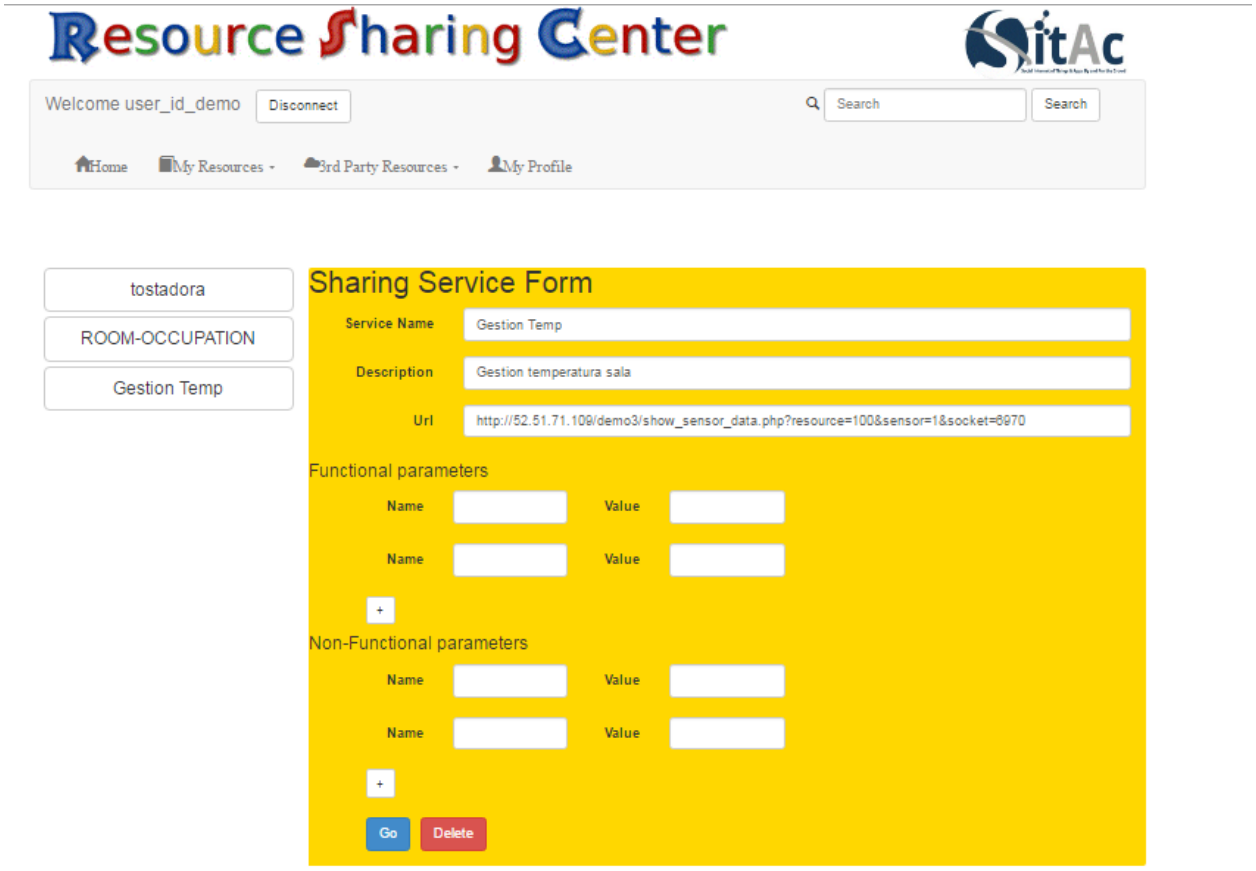

Figura 64: Inventario de servicios de terceros

3.3.2.9 My profile

La visión del perfil está organizada de forma uniforme con el resto de la página, esto es, a la izquierda los botones con las funcionalidades que puedas tener, y a la derecha la información que muestran estas funcionalidades.

Esta parte de la aplicación tiene tres funcionalidades:

- Consultar o actualizar el perfil del usuario (Figura 65): En la parte derecha están las 10 palabras clave más importantes que representan el perfil junto con el peso de cada una.

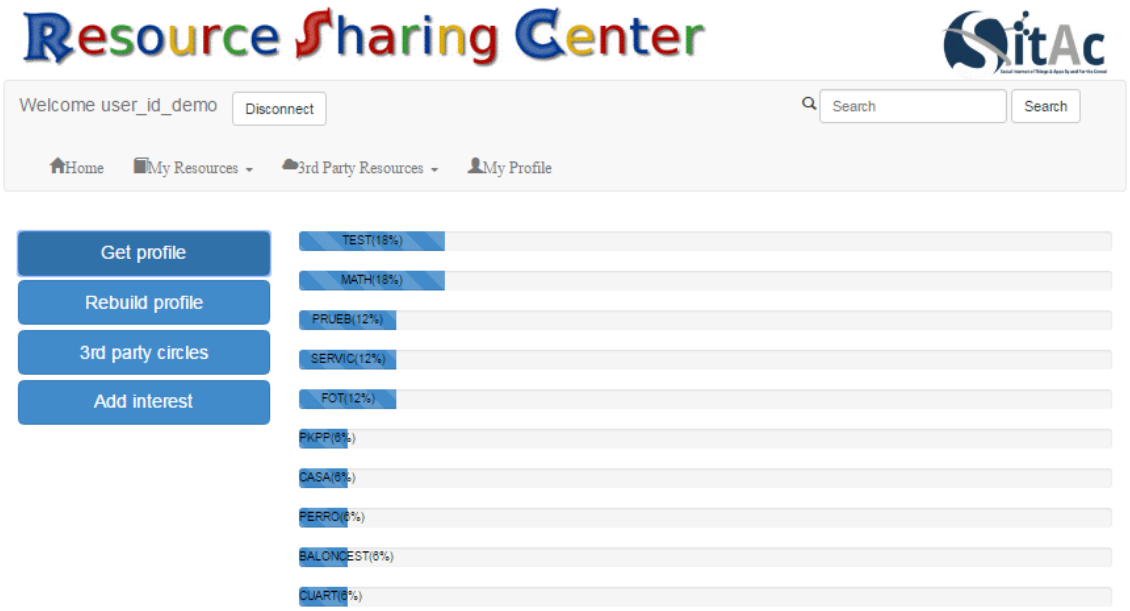

Figura 65: Perfil del usuario

- Consulta de círculos de terceros a los que se pertenece: Se carga en la parte derecha una tabla con una lista de círculos a los que perteneces.

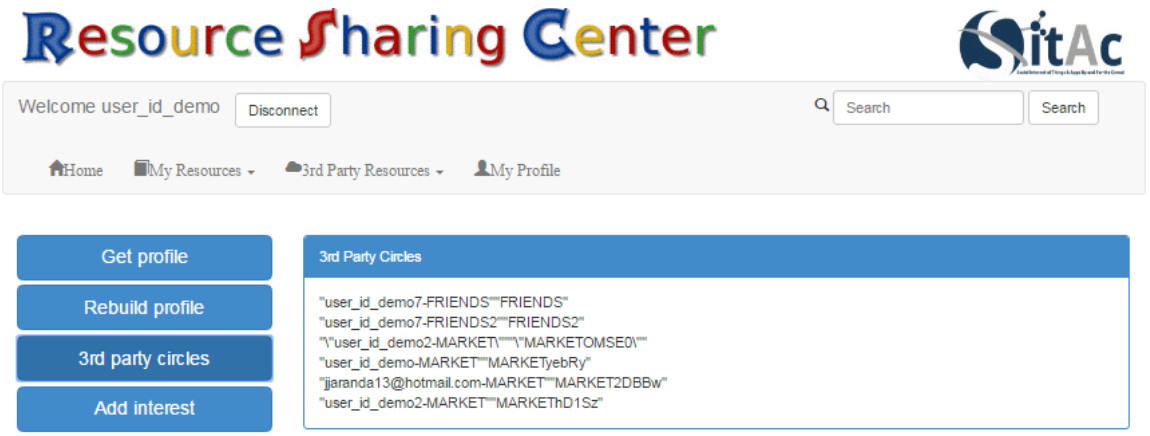

Figura 66: Círculos de terceros

- Añadir un interés (Figura 67): Se muestra la ventana modal de compartición, pero sólo con el campo de descripción, para que pongas un interés que quieras que se agregue a tu perfil.

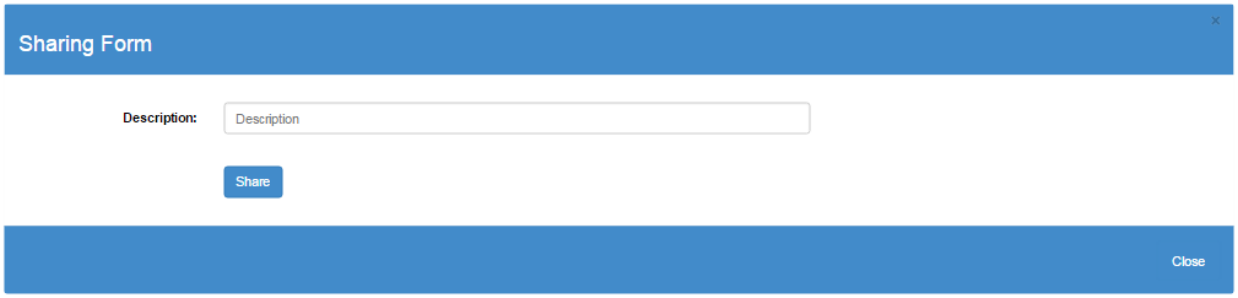

Figura 67: Añadir interés (Formulario de compartición)

3.3.2.10 Búsquedas

Las búsquedas se implementan aplicando el algoritmo lematizador a la entrada que se introduce en el cuadro de búsqueda, y buscando esos lemas en la tabla MMS_RESOURCE_TABLE, con la respuesta de esa búsqueda se genera una tabla en la que aparecen los elementos que se han encontrado junto con información sobre su precio, un botón para poder ver el recurso si este es una carpeta publica, y un botón para hacer pick up del recurso, esto es introducirlo en tu inventario de terceros, como se muestra en la Figura 68.

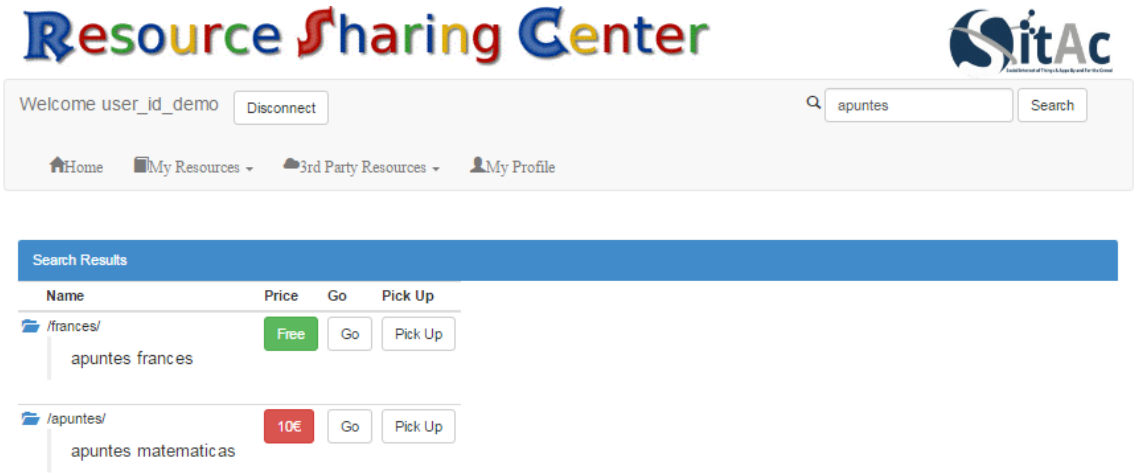

Figura 68: Resultados de una búsqueda

Al hacer pick up de un contenido público, automáticamente se añade al inventario propio, con el correspondiente mensaje de información. Si el servicio es de pago, aparecerá una simulación del pago (ya que no hay interacción con paypal u otros servicios de pago) para poder obtener el contenido. Y si no perteneces a los círculos en el que se puede obtener el recurso, se te ofrecerá la opción de pedir permiso para acceder a ese círculo, según se muestra en la Figura 69.

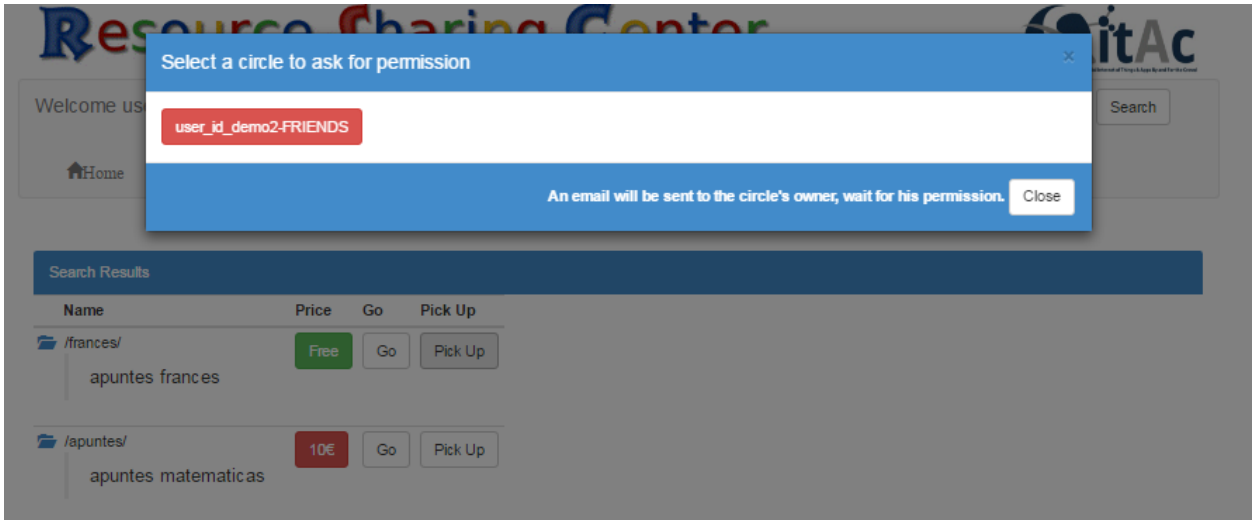

Figura 69: Pedir permiso para acceder a un círculo.

3.4 Otros elementos

En esta sección se describen otras funcionalidades del RSC que complementan a lo ya descrito.

3.4.1 Algoritmo MMS Rank

El MMS Rank es una aportación original del Resource Sharing Center, es una medida entre la novedad de un recurso y la popularidad, que se usa para ordenar los resultados de las búsquedas.

Por analogía con el *Page Rank* de Google [64] se ha llamado MMS Rank

Mms_rank = log (k1* (marca de tiempo en milisegundos)) + log (k2 * contador de pickup)

El criterio escogido para asignar los valores de k1 y de k2 ha sido no priorizar una variable sobre la otra. Se consideran el tiempo y los *pickups* para determinar la pareja de constantes k1 y k2 que permita igualar la importancia de las dos partes de la ecuación.

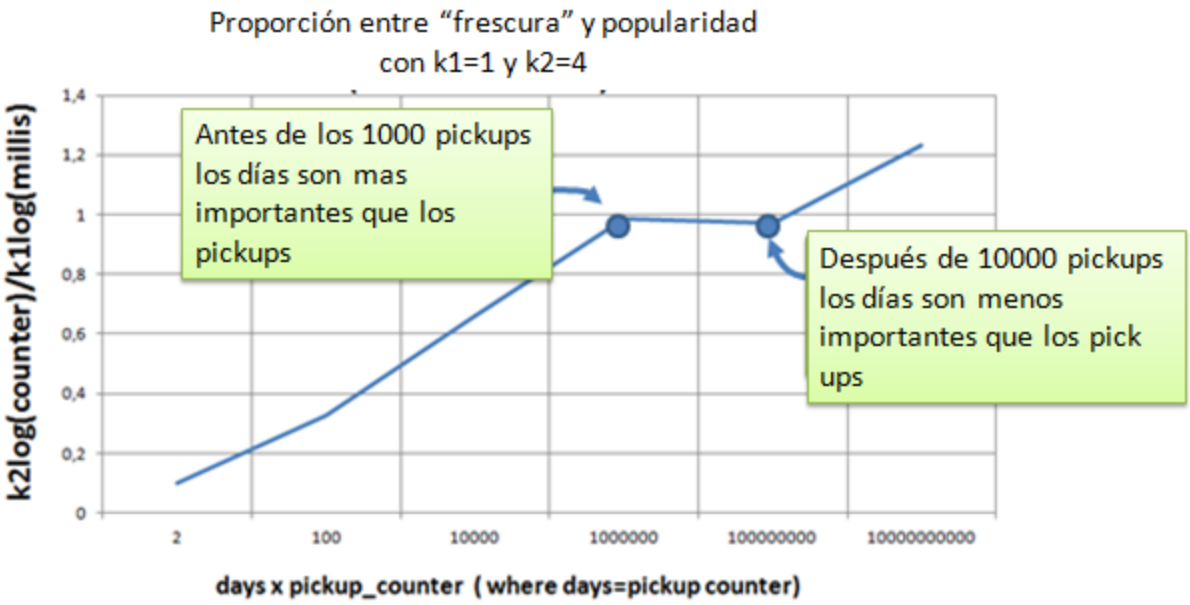

Figura 70: Cálculo de constantes de MMS Rank. Proporción entre frescura y popularidad

Como se ve en la Figura 70, con k1=1 y k2=4, la importancia del tiempo es más importante antes de los 1000 pickups, pero después de los 10000 pickups el contenido es lo suficientemente popular para que los pickups tengan más peso.

4 Módulo Analytics

El objetivo del módulo Analytics es recomendar a los usuarios contenidos basados en sus perfiles. Para conseguir esto, los usuarios serán agrupados en clusters y la recomendación para un usuario específico se basará en las claves que pertenecen a otros usuarios ubicados en el mismo cluster. Este proceso se representa en la Figura 71.

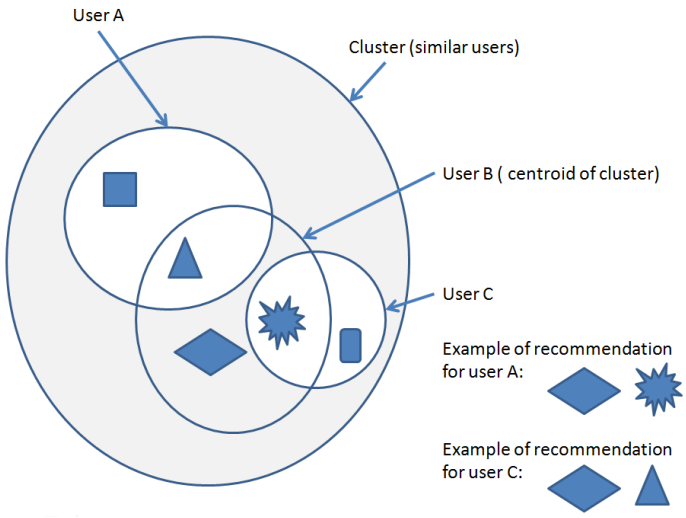

Figura 71: Recomendación y cluster para un usuario

4.1 Definición del clúster

En esta sección se describen los principales parámetros que definirán el proceso de clustering sobre los datos que proporciona el Resource Sharing Center (RSC).

Para agrupar en clusters a los distintos usuarios hace falta una función distancia, es decir que cumpla estas cuatro propiedades:

1. $d(x, y) \geq 0$ (no negatividad)
2. $d(x, y) = 0$ si y solo si $x = y$ (identidad indiscernible)
3. $d(x, y) = d(y, x)$ (simetría)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (desigualdad triangular)

Esta función distancia será usada con el fin de generar clusters de usuarios muy similares, siendo muy distintos los clusters entre ellos como se ve en la Figura 72.

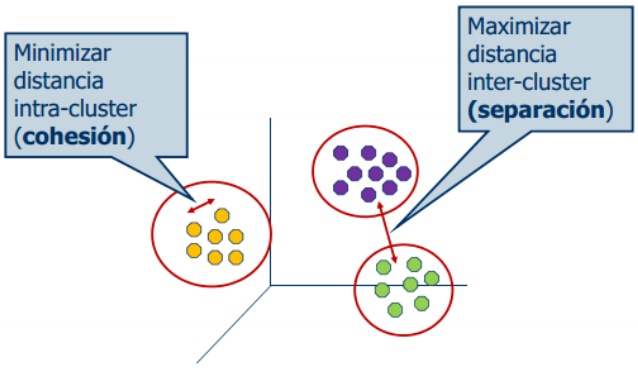

Figura 72: Objetivo del proceso de clustering

4.1.1 Normalización de los datos

Los datos sobre los que se desarrolla el proceso de clustering serán los perfiles de los usuarios del RSC, por lo que es necesario formatear esos datos para establecer una función distancia correcta sobre estos.

El formato de los perfiles de usuarios es el siguiente:

[{"key":"MUSIC","weight":2},

{"key":"SHAKIRA","weight":1},

{"key":"LENNON","weight":1},

{"key":"COMPUTERS","weight":1}]

Para la función distancia se consideran las diez primeras parejas *(key, weight)* de cada usuario normalizando los pesos de estas:

Peso_total= peso(key0)+…+peso(key1)

Peso_normalizado_i= peso(key i) / Peso_total (para i de 1 a 10)

4.1.2 Distancia de una clave

Para cada clave de un usuario A, si esa clave no está presente en el usuario B, entonces la distancia de esta clave es el peso (*weight*) en el usuario A. Esto hace que cuanto más peso tenga esa clave en el usuario A, implicará más distancia respecto al usuario B.

$$\text{Dist(i)} = weight_A(i)$$

Si la clave existe en ambos usuarios, los pesos son comparados y la distancia es:

$$dist_{AB}(i) = |weight_A(i) - weight_B(i)|$$

Este tipo de distancia se corresponde con la distancia Manhattan.

4.1.3 Distancia entre usuarios

La distancia total entre dos usuarios considera la distancia del conjunto de claves compuesto por la unión de las claves del usuario A y las del usuario B.

$$dist\ (A,B) = \sum_{i=1}^{i=N} dist_{AB}(i)$$

Esta función cumple con las cuatro propiedades de la función distancia.

4.2 Selección de algoritmo y librería

El algoritmo seleccionado para el proceso de clustering sobre los perfiles del RSC es el algoritmo K-means, pudiendo resolver el problema del número de clusters sobre el que operar para reducir la complejidad gracias a la métrica SSE.

Los algoritmos basados en densidad se descartan porque son útiles para conjuntos de datos de alta y baja densidad, pero no para los datos con que se obtienen del RSC habrá una gran cantidad de datos dispersos ya que son claves que tendrán muy pocos usuarios haciendo que aumente el número de clusters.

Los algoritmos jerárquicos también son descartados debido a su baja escalabilidad, ya que este proceso está orientado a tratar con cantidades masivas de datos.

4.2.1.1 Criterio de selección de la librería

Las principales características que debe tener la librería que ejecute el clustering son:

- Debe ser desplegable en Amazon.
- Debe implementar K-Means.
- Fácil de usar y aprender.
- La función distancia puede ser definida por el usuario.

- Con capacidad de manejar un número muy grande (del orden de millones) de puntos (usuarios).

La librería Cluster 1.2.2 cumple estos requerimientos y está disponible para su descarga [65].

4.2.1.2 Funcionamiento y descripción de la librería

La librería consta de una carpeta con tres ficheros y una sub carpeta:

- Cluster.py: Definición del cluster y funciones sobre él.
- Utils.py: Con funciones auxiliares.
- Matrix.py: Representación de los ítems auxiliar.
- Subcarpeta Method: Con la implementación de distintos tipos de clustering
    - Kmeans.py
    - Hierarchical.py
    - Base.py

Las principales clases y funciones de kmeans.py son:

- class KMeansClustering(object): Implementación del método de clustering KMeans.
- getclusters(self, count): Genera *count* clusters. Esta es la función principal, inicializa los clusters y los mueve en función de las distancias a los centroides.
- assign_item(self, item, origin): Asigna un *ítem* de un cluster dado hacia el cluster más cercano.
- move_item(self, item, origin, destination): Mueve un *ítem* de un cluster hacia otro cluster.
- initialise_clusters(self, input_, clustercount): Inicializa los clusters distribuyendo los *ítems* en los k clusters.

4.3 Diseño del módulo analytics

El módulo analytics consta de dos partes:

- Módulo "batch" para realizar el proceso de clustering (calcular los clusters y centroides).
- Una función de recomendación basada en una búsqueda usando las claves del centroide al que pertenece el usuario y él no posee.

4.3.1 Población de datos

El primer paso para realizar el clustering es rellenar el conjunto de datos con usuarios generados basados en el comportamiento real de estos.

Para aproximarse al comportamiento real en que los humanos no usan más de 1000 palabras en general, y la mayoría de estas son sobre un conjunto de 300 [66] [67], por lo tanto se generarán usuarios aleatorios basados en este comportamiento, es decir, el 80 por ciento de las palabras que usen, será de entre 300 posibles, y el 20 por ciento serán de entre el resto.

4.3.2 Pasos de proceso

En esta sección se describen los pasos que se siguen para realizar el proceso de clustering y recomendación.

4.3.2.1 Módulo batch

El proceso que se sigue para crear el fichero batch que se ejecutará periódicamente para realizar el proceso de clustering sobre los perfiles es el siguiente:

1) Creación de una lista de perfiles simulados: Para construir cada cada perfil se tienen en cuenta que la gente normalmente no usa más de un rango de entre 300 y 1000 palabras [68].

   Por lo tanto para la construcción de los perfiles se tendrá en cuenta, un diccionario de 1000 palabras, compuesto de 300 palabras populares y 700 menos populares. Las palabras serán cadenas numéricas como “123”, por lo que las palabras populares serán las cadenas que representen números menores que 300.

   Los perfiles están compuestos de 10 palabras, de las cuales 8 se asignan aleatoriamente de entre las primeras 300, y 2 palabras se asignan aleatoriamente de entre las 700 palabras menos populares.

   Estos perfiles simulados están adaptados al uso común del lenguaje y el peso de cada palabra en el perfil será un número aleatorio manteniendo siempre que la suma de los pesos de un perfil será 1.

2) Rellenar un array con los perfiles: Este array puede ser rellenado con perfiles reales o con perfiles simulados como los descritos en el punto 1. El objetivo es realizar el proceso de clustering para determinar el número óptimo de clusters que funciona como parámetro en el algoritmo K-Means. Para determinar este número se usaran los perfiles simulados, una vez se sepa el número de clusters óptimos para el conjunto de datos no hará falta trabajar con perfiles simulados.

3) Normalizar los pesos de los perfiles de usuarios: Esto no hace falta para los perfiles simulados, ya que los pesos siempre sumarán 1, pero los perfiles reales pueden ser usuarios nuevos con menos, o más de 10 palabras.

4) Implementación del proceso de clustering: En pseudocódigo

   For "i" in USER_PROFILE_TABLE

   Lista.append(Normaliza(readUser(i)))

   Next

   cl = KMeansClustering(lista,mifunciondistancia)

   misclusters=cl.getclusters(NUMERO)

   En este proceso también se realiza:

   - La implementación de la función distancia definida con anterioridad,
   - El cálculo interno en la función getclusters() del centroide de cada cluster, como un perfil teórico compuesto por las palabras más populares del cluster al que representa.

5) Ejecutar el proceso de clustering para distintos números de clusters y calcular la métrica SSE en cada caso, de esta forma se puede identificar el número óptimo de clusters necesario, que se representa en el codo de la gráfica de la Figura 73.

   Además de esto hay que asegurarse de que todos los usuarios tengan al menos una palabra en común con su centroide, ya que en caso contrario hay que generar más clusters y centroides más pequeños, y que los clusters que contienen un solo perfil deben ser menos del 5% del total.

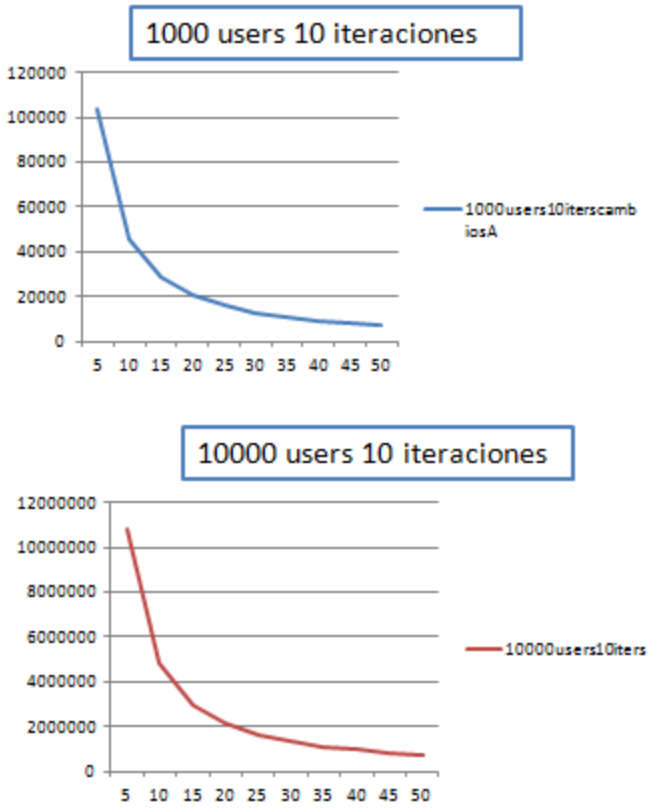

Figura 73: Calculo de SSE para distintos números de cluster

Como se ve en la Figura 62 el número óptimo de clusters es entre 20 y 25, este cálculo se ha realizado para distintos valores del número de usuarios.

Y dando el siguiente resultado de control de que no haya clusters vacíos, ni clusters que no tengan palabras en común con su centroide, esto es, que la distancia de los usuarios y sus centroides no sea mayor que 2, por nuestra definición de la distancia.

sses:[10834540.620368855, 4802554.054808634, 2959546.554473516, 2116794.84479742, 1653088.636397708, 1333936.1916187287, 1108745.7178465202, 945072.4200875383, 842581.4578284224, 741133.7106416145]

distancias >=2: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

vacíos: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Para futuras ejecuciones con perfiles reales se usaran 25 clusters para representar los perfiles y se usarán sus centroides para realizar las recomendaciones.

El fichero batch ejecutará el clustering sobre los perfiles reales y almacenará los centroides en la tabla CENTROIDS de SimpleDB.

4.3.2.2 Recomendador interactivo

Esta parte está compuesta por la invocación del cliente de la función recommendMe() del servidor.

Cuando un usuario hace login en el RSC, se invoca la función recommendMe(), que realiza los siguientes pasos:

1. Calcula la distancia del usuario actual con todos los centroides para determinar cuál es el más cercano.
2. Elige de forma aleatoria una palabra del centroide.
3. Realiza una búsqueda basada en esa palabra en la tabla MMS_RESOURCE_TABLE.
4. Elige al azar un ítem de la lista de resultados.
5. Ofrece como recomendación el ítem seleccionado.

4.4 API del módulo Analitycs

En esta sección se describen los principales ficheros y funciones con los que se ha trabajado para realizar clustering sobre los perfiles de usuarios del RSC.

4.4.1 Adiciones a la librería

A la librería original se le han añadido las siguientes funcionalidades para que se adapte al tipo específico de datos del RSC [69].

En kmeans.py

- HDgetclusters(self, count, max_iterations): Similar a la función original getClusters, pero incorporando un límite a las iteraciones que se producen por cambios mínimos en los centroides. Además de esto, para reducir la complejidad de la función no se recalculan los centroides cada vez que se asigna un perfil a un cluster.
- HDassign_item(self, item, origin, origin_centroid, my_centroids): Igual que la función assign_item, pero gestionando también los centroides y reduciendo así la complejidad.

En utils.py

- HDcentroid(data): Adapta la gestión de los centroides al tipo de datos del RSC.

Se ha creado el nuevo fichero:

- HDdistances.py: Con el cálculo de la distancia entre usuarios, y el cálculo de la métrica SSE. Con las siguientes funciones:
    - HDdistItems(profile1,profile2): Definición de la función distancia entre usuarios.
    - HDcomputeSSE(solution,numclusters): Esta métrica mide la cohesión de los usuarios en un cluster y la separación entre clusters.
    - HDcomputeSSEfromCentroids(items,centroids): Similar a la función anterior, pero midiendo las distancias entre centroides.

4.4.2 Módulo batch

Las funciones que se definen ahora están definidas en el fichero RealClustering.py

- getProfiles(): Conexión con SimpleDB y búsqueda de los perfiles.
- distUsers(perfil1,perfil2): Función distancia entre los perfiles.
- guardaCentroides(listaCentroides,timeStamp):Tras el proceso de clustering almacena los centroides en la tabla CENTROIDS.
- Main(): Realiza el proceso de clustering de la librería, con las funciones descritas anteriormente.

4.4.3 Interacción con el RSC

En el módulo Hello del RSC, hay una serie de funciones escritas en Python para realizar la consulta de los centroides y ofrecer una recomendación.

- getKeyword(user_id): Compara el perfil del usuarios con los perfiles almacenados en la tabla centroides, para saber a qué cluster pertenece, y ofrecerte una palabra al azar perteneciente a es cluster.
- getRecomendedKeyword(): Esta función llama a la anterior, pasándole como parámetro el user_id del usuario actual para buscar su perfil.

En el módulo cliente se ha definido la función:

- recommendOne(): Que realiza una llamada AJAX a getRecomendedKeyword() para construir dinámicamente en el cliente una tabla con la recomendación, y las opciones de verlo y hacer pick up.

4.4.4 Pruebas de la solución

Para hacer pruebas de la solución escogida se han creado los siguientes ficheros, para generar usuarios de forma aleatoria, y hacer pruebas de clustering sobre ellos. Los ficheros son los siguientes:

- HDExample.py: Fichero con las funciones necesarias para realizar el proceso de clustering y calcular la métrica SSE para validar el número de clusters con el que se realizará este proceso.
- HDexample - Only Users.py: En este fichero se encuentran las funciones necesarias para generar perfiles de usuarios ficticios sobre los que hacer pruebas.
- HDCentroidesCercanos_new.py: En este fichero se encuentran las funciones necesarias para las comprobaciones de que los centroides están bien elegidos, esto es, que los usuarios tienen al menos una palabra en común con el centroide y no hay más de un 5% de clusters de un solo elemento.

4.5 Modelo de datos

En esta sección se describirán los tipos de datos que hay en las bases de datos, la forma en que se relacionan y las condiciones que deben cumplir los mismos.

4.5.1 Dominio CENTROIDS

Tipo de campo	Nombre del campo	Comentario
Nombre del objeto	Cluster_id	
Atributo	Profile	Lista de (clave,peso)

Donde cluster_id= Marca de tiempo de la última ejecución del clustering + cluster_number

Cluster_number es un numero de 1 a N (donde N= número de clusters)

En cada ejecución esta table es borrada y rellenada.

4.6 Cambios en la librería

Para solucionar distintos problemas para los que la librería cluster 1.2.2 no estaba preparada se han realizado los siguientes cambios:

- Se ha reducido la complejidad computacional permitiendo ahorrar en la implementación de K-Means el cálculo redundante del centroide cada vez que se asigna un perfil a un cluster.
- Se ha limitado el número de iteraciones que realiza el algoritmo K-Means limitando también de esta forma el tiempo de computación.
- La librería no estaba diseñada para el problema de la alta dimensionalidad que presentan los perfiles de usuario del RSC ya que 1000 palabras son 1000 dimensiones.
- Para calcular los centroides la librería usaba la mediana, que implica una ordenación mediante el algoritmo quicksort pero para los datos del RSC también se puede usar la media de los datos, reduciendo la complejidad computacional.

4.6.1 Reducción de la complejidad

En la implementación de la librería cluster 1.2.2, al realizar clustering con el algoritmo K-Means se recalculan los centroides cada vez que se asigna un *ítem* a un cluster, ya que tiene que recalcular cual es el centroide más cercano, suponiendo un gran aumento en la complejidad computacional, esto se ha resuelto teniendo pre calculados en un *diccionario* formado por la composición de clusters y sus centroides que permite ahorrar ese cálculo del centroide al que se llama la mayoría de las veces. Como se ve en la Figura 74 la complejidad se ha reducido de orden(n^2) a orden(n)

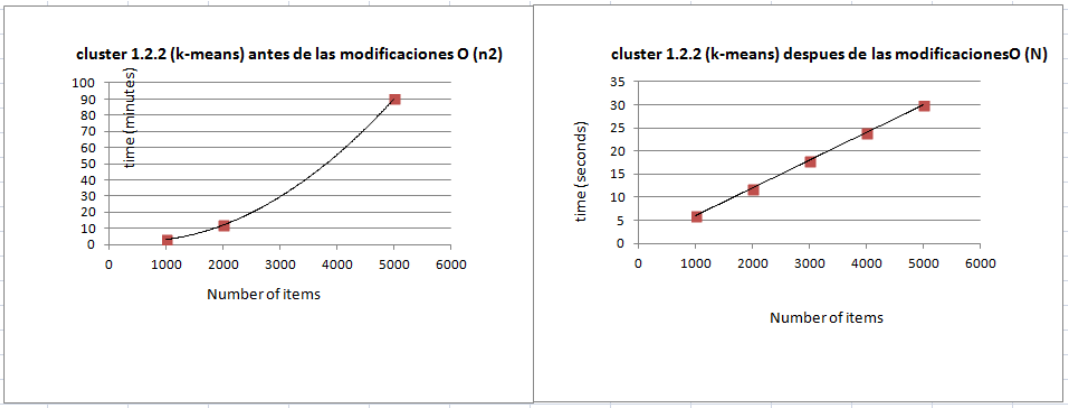

Figura 74: Reducción de la complejidad computacional

4.6.2 Optimización del tiempo de computación

Para reducir el tiempo de computación del algoritmo K-Means al igual que otras implementaciones como la de la librería SciKitLearn [70] se ha limitado el número de iteraciones en las que el algoritmo K-Means sigue recalculando los elementos de cada cluster, ya que a partir de las 10 iteraciones como se ve en la Figura 75 los cambios en la métrica SSE son mínimos y no afectan a la solución.

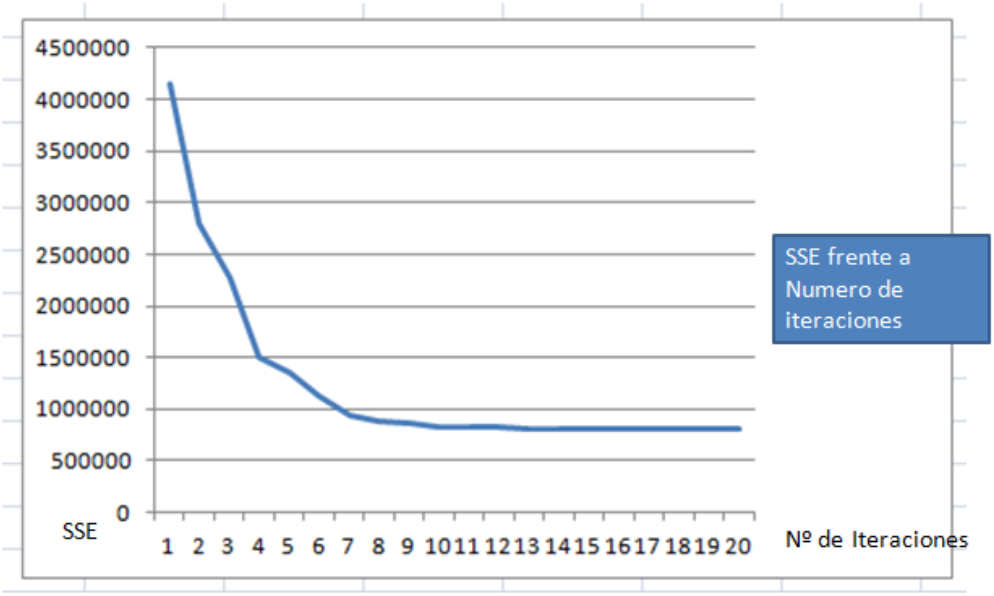

Figura 75: Variación de la métrica SSE frente a número de iteraciones

4.6.3 Problema de la alta dimensionalidad

Debido a la alta dimensionalidad de los perfiles del RSC, se han tenido que redefinir las funciones de cálculo de centroides y la función distancia para que puedan manejar estos datos más complejos que los datos para los que estaba preparada la librería.

4.6.4 Validación del uso de la media en lugar de la mediana

Para optimizar el tiempo de ejecución del proceso de clustering se calcula el centroide como la media de los elementos del cluster en lugar de como la mediana, la distribución de las palabras

en los perfiles del RSC permite esto, como se puede ver en la Figura 76 en la que se muestra que el cálculo de la métrica SSE es prácticamente igual para el proceso con mediana y con media.

	number of clusters									
	5	10	15	20	25	30	35	40	45	50
SSE median	1637	758	480	316	269	232	192	150	138	119
SSE mean	1702	772	489	365	288	214	203	152	136	110

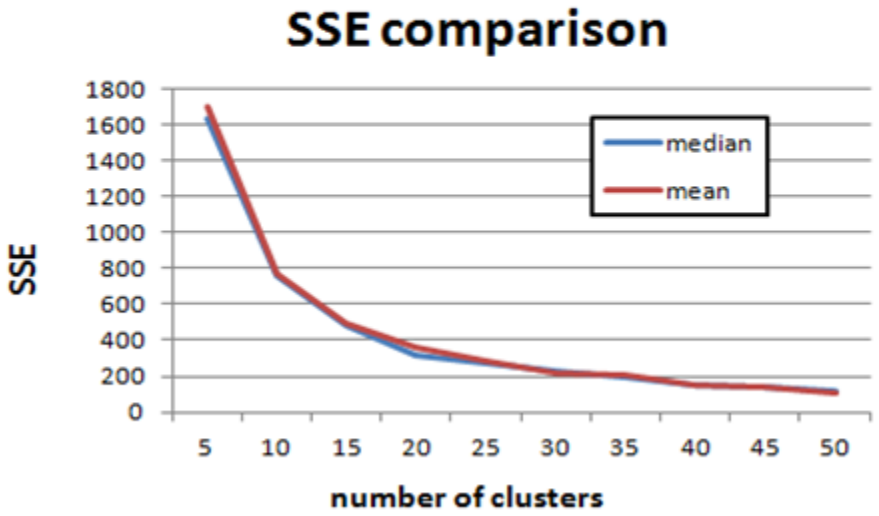

Figura 76: Comparación del SSE calculado usando la media y la mediana

5 Resultados

Los resultados presentados muestran la consecución de los objetivos planteados, se ha logrado realizar un servicio web escalable orientado a la compartición y a un uso masivo que está desplegada en Amazon siendo completamente funcional. A parte de esto se ha desarrollado también un estudio sobre el clustering de alta dimensionalidad que derivará en la presentación de un artículo académico sobre la estimación del tamaño de los sets de datos para realizar clustering fiable sobre ellos.

5.1 Servicio Resource Sharing Center

Se ha implementado un servicio web orientado a la compartición de recursos, ya sean gratuitos o de pago, como se ve en la Figura 77

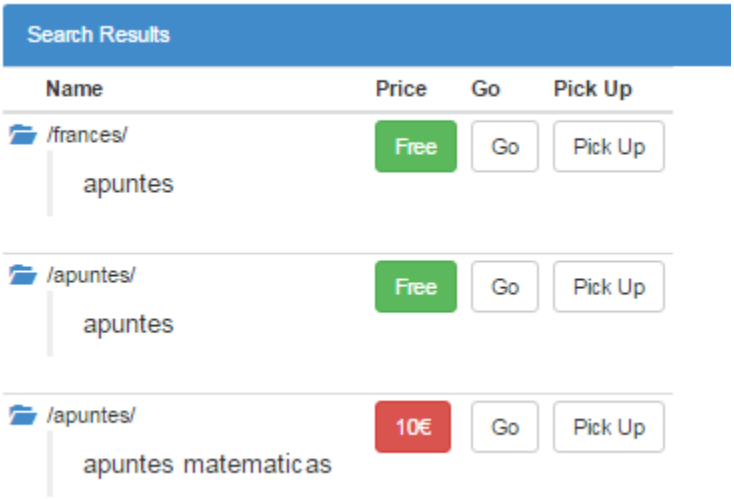

Figura 77: Recursos compartidos gratuitos y de pago

Los recursos a compartir pueden ser de varios tipos, tanto recursos lógicos (ficheros, carpetas), como físicos (impresoras compartidas, coche con url de blablacar).

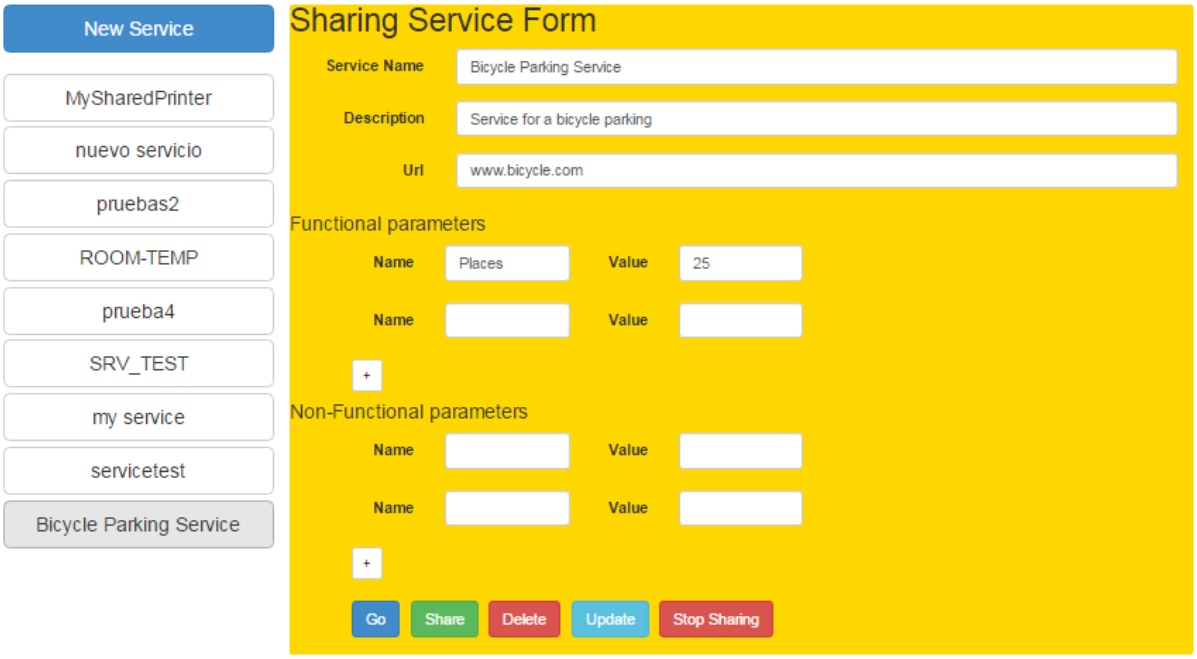

Figura 78: Servicio para compartir "parking de bicis"

Los usuarios comparten los recursos entre ellos, en el inventario de terceros de la aplicación se pueden ver los recursos que se tienen de otros usuarios, y en el inventario propio se pueden ver los recursos que se han compartido.

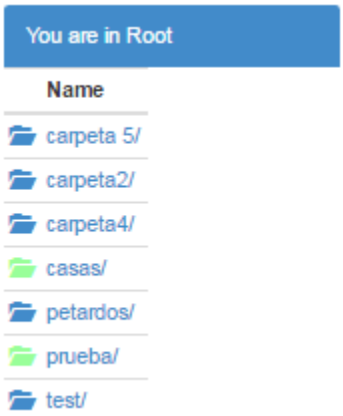

Figura 79: Carpetas de usuario, el icono verde para las carpetas compartidas.

Se pueden implementar búsquedas sobre los servicios compartidos, como se ve e la figura 80.

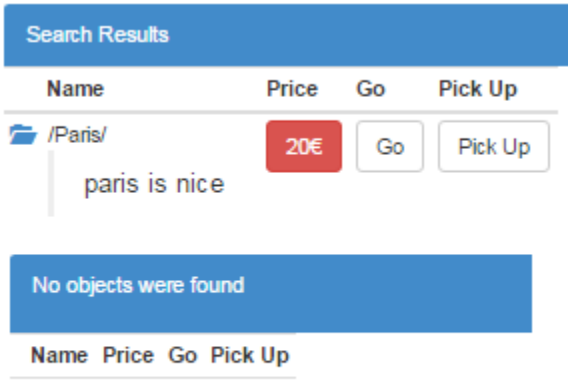

Figura 80: Posibles resultados de las búsquedas

Además se ha implementado la construcción de un perfil de usuario (Figura 81) en función de los intereses que él añada, o los recursos de terceros que incorpore a su inventario.

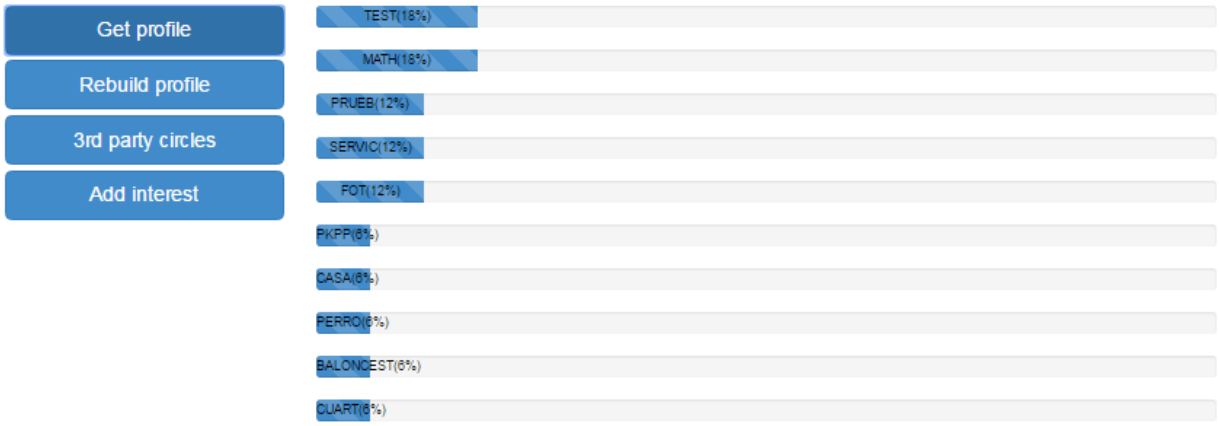

Figura 81: Perfil de usuario del RSC

También se ha desarrollado el motor de recomendaciones basado en clustering sobre big data y se hacen recomendaciones al usuario en función del cluster en que se encuentre.

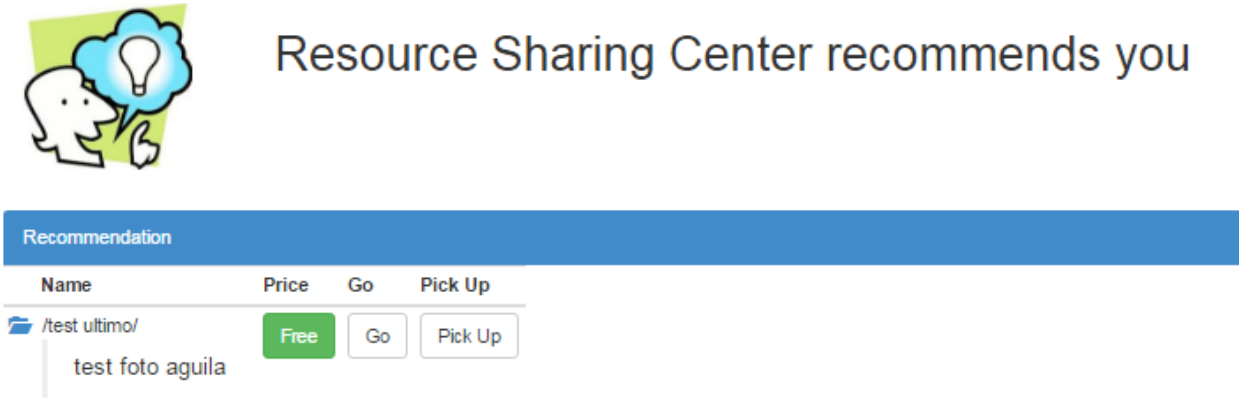

Figura 82: Recomendación basada en los gustos del usuario

5.2 Servicio web escalable

Se ha estudiado la escalabilidad del servicio desplegado en Amazon en una maquina micro.t1 para permitir un ahorro en los costes (ya que micro.t1 es la máquina más barata de AWS) y establecer el umbral en el que Elastic Beanstalk instanciará una máquina micro.t1 del servicio EC2 nueva (mediante un disparador de elasticidad que se puede configurar), y gracias a la herramienta Apache JMeter se han obtenido los siguientes resultados.

- Para operaciones de lectura:

  Los tiempos de respuesta rondan los 40 ms.

  La máquina micro.t1 no puede procesar más de 5000 peticiones por minuto, cuando se acerca a este límite la latencia se incrementa hasta llegar a ser mayor de 1 segundo.

  Asumiendo el rendimiento como una métrica de concurrencia, para un 5% de concurrencia en la máquina micro.t1, que tiene capacidad para 100.000 usuarios.

$$\frac{5000}{0.05} = 100.000\ (usuarios)$$

- Para N operaciones de lectura y 1 de escritura

  Los tiempos de respuesta son de 40ms, y la máxima concurrencia que puede procesar la máquina micro.t1 es de 3000 peticiones por minuto, por lo que tiene capacidad para 60.000 usuarios

$$\frac{3000}{0.05} = 60.000\ (usuarios)$$

Para el RSC se asume que una máquina micro.t1 soporta 50.000 usuarios sin problemas en los escenarios asumiendo una concurrencia del 5%. La media del tiempo de respuesta en estas condiciones está por debajo de los 100 ms.

5.3 Despliegue en Amazon

El servicio es funcional y está desplegado en Amazon Web Service en la siguiente url:

http://mms-pyenv3.eu-west-1.elasticbeanstalk.com/

Es una aplicación basada en el modelo PaaS que en Amazon se implementa mediante el servicio de Elastic Beanstalk, en este caso con dos entornos siguiendo la arquitectura que ya se ha explicado en esta memoria.

Para la implementación del back-end se ha usado un módulo en Node.js, que permite llamadas asíncronas para comunicarse con las tablas de la base de datos del servicio de Amazon SimpleDB, una base de datos NoSQL, que ofrece una gran eficiencia, sin la que no sería posible el big data.

Además el front-end lo ofrece una parte servidora en Python, que se encarga de gestionar el acceso y brindar la página al usuario, y para minimizar los costes y el tráfico con los servidores, es el cliente el que realiza las llamadas directamente al servicio de almacenamiento masivo de Amazon (S3).

El front-end está desarrollado usando HTML5, JavaScript, JQuery y AJAX, ya que mediante la tecnología AJAX se realizan todas las llamadas al back-end y mediante JavaScript y JQuery se construye la página dinámicamente en función de las respuestas recibidas.

Para el aspecto gráfico de la página cliente se ha usado Twitter Bootstrap, que ofrece un diseño limpio y estructurado a la vez que permite decidir cómo se ve el servicio en función del tamaño del dispositivo en el que se vea.

5.3.1 Sobre la implementación del servicio

Durante el desarrollo del servicio han surgido necesidades de nuevas que no estaban especificadas en un principio, como los desarrollos que se muestran a continuación:

- Ha sido necesario el estudio e implementación de tablas basadas en índice inverso para mejorar la eficiencia de las búsquedas del servicio que se basan en un parámetro de las tablas, esto es posible gracias a la eficiencia que ofrecen las bases de datos NoSQL.
- Para la ordenación de los elementos compartidos y la muestra de los mismos se ha generado la medida MMSRank, llamada así por analogía con el PageRank de Google, el MMSRank mide la popularidad de un recurso en función de la antigüedad del mismo, para que en las búsquedas aparezcan primero las opciones más populares y nuevas.
- También se ha estudiado y desarrollado la generación de *thumbnails* en el lado cliente, de forma transparente para el usuario, ya que es un proceso que se ejecuta automáticamente cuando este sube una imagen al servicio. Este servicio permite ahorrar la carga de imágenes que pueden ser muy pesadas, ya que los usuarios pueden ver los contenidos de sus carpetas cada vez que acceden a su inventario de recursos.
- Se ha usado un algoritmo lematizador para permitir obtener las palabras claves de descripciones de recursos, que son los lemas de las palabras usadas, para que en un recurso o interés descrito por un lema, se tengan en cuenta todas las palabras derivadas del mismo.
- Se permite la autenticación delegada usando el protocolo OpenId Connect, que permite que se acceda a la aplicación con la cuenta de Google.

5.4 Estudio sobre clustering y mejora de la librería cluster 1.2.2

Durante el desarrollo de este trabajo se ha realizado un estudio sobre clustering para clasificación de perfiles de usuario que ha resultado más complejo de lo previsto, ya que no se habían contemplado los problemas derivados de la alta dimensionalidad de los datos y la adaptación de las librerías disponibles a este tipo de datos.

Se ha realizado un estudio sobre el uso del lenguaje habitual por parte del ser humano, y sobre este uso del lenguaje (en base a las 300 palabras que de media se usan con más frecuencia de un total de no más de 1000) se han realizado las pruebas de clustering sobre los perfiles de usuarios y se ha comprobado que los usuarios se pueden clasificar en 25 clusters de forma efectiva sin dejar usuarios sin representar, gracias a la métrica SSE.

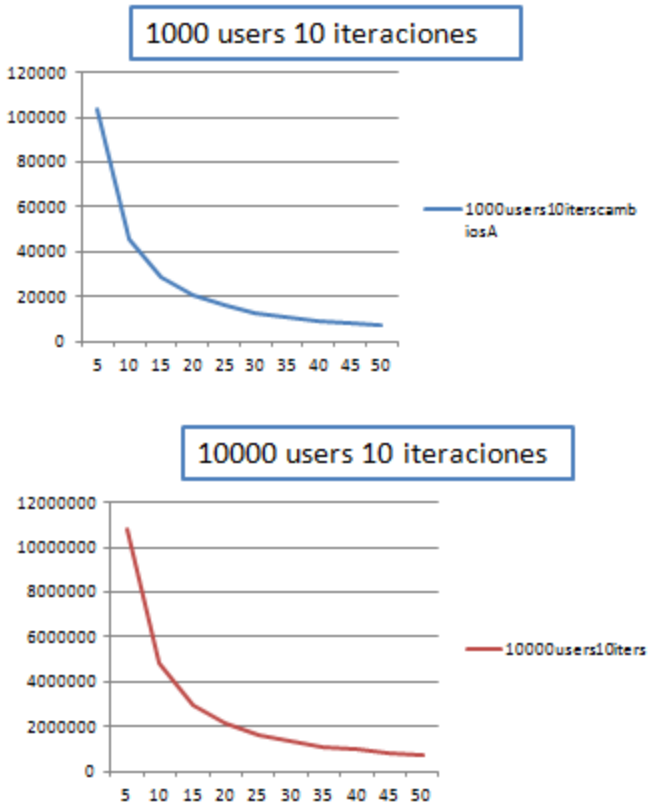

Figura 83: Análisis de la métrica SSE

Se ha mejorado la librería cluster 1.2.2 para adaptarla a la alta dimensionalidad y a nuestro tipo de datos. Estos cambios han sido, reducción de la complejidad, límite de iteraciones en el cálculo del cluster, inclusión de nuevas funciones que permiten el clustering de alta dimensionalidad, nueva función distancia.

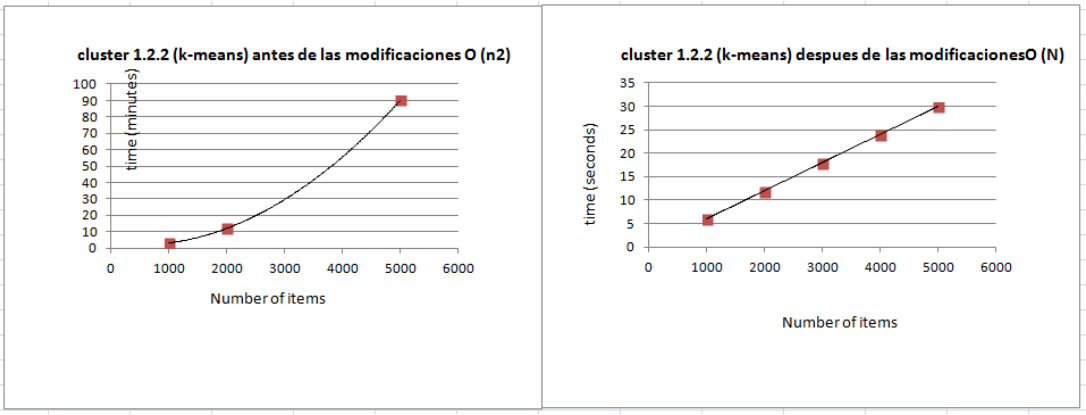

Figura 84: Reducción de la complejidad de la librería cluster 1.2.2

Durante el estudio del clustering para big data se ha comprobado que se llega un punto en el que el número de usuarios es tan grande que deja de afectar la cantidad de usuarios, esto es, se puede usar un número menor de usuarios y extrapolar los resultados sobre ellos a toda la población.

Siendo c1 un conjunto de centroides con un número de elementos n1 y c2 otro conjunto de centroides con un número de elementos n2, la distancia entre c1 y c2 se define como:

$$dist = \sum_{i=1}^{25} \min\{C_{1i} - C_{2j}, j \in (1..25)\}$$

Esta distancia tiende a cero cuando el número de datos con los que se obtienen los centroides aumenta como se ve en la figura 85, por lo que no hay necesidad de realizar el proceso de clustering para un número muy grande de datos, sino que hay que realizarlo para el numero N en el que deja de aumentar esta distancia.

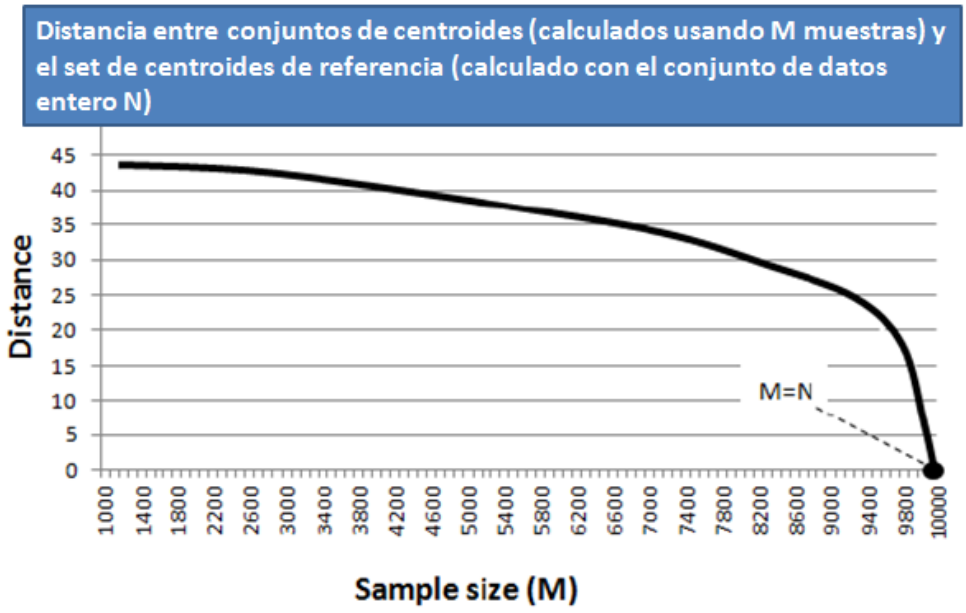

Figura 85: Calculo de distancias entre centroides

Este estudio sobre el agrupamiento ha dado lugar a un artículo sobre el tamaño del conjunto de datos para realizar clustering fiable sobre el mismo (An effective coreset size estimation for K-clustering algorithms and large high dimensional datasets), que pretendemos enviar a la revista Sort journal [71]

A continuación se muestra el abstract del mismo:

“Big Data pose new computational challenges for understanding huge volumes of heterogeneous data. That is, in recent years high data rate applications have become more and more popular. Many cyber-physical systems, including the Internet of Things (IoT) applications, have started generating huge amount of data, from heterogeneous sources, arriving at a high speed and characterized by high dimensionality feature. High dimensionality introduces spurious correlations, noise accumulation, and computational costs difficult to afford. These unique features imply that statistical procedures should be designed or adapted taking into account the aforementioned issues. High dimensionality problems are more sensitive, in terms of computational costs, to the growth of the size of datasets particularly produced by the large_scale IoT applications. This often causes that non-lineal data mining procedures are directly discarded. However, even linear procedures such as K-Means clustering, become unaffordable when huge datasets are faced. In this paper, we present an strategy for solving clustering problems of big and high dimensional datasets, by sizing a coreset (sub-sampled dataset), in order to save computational costs and afford the problem in a reasonable time. The motivation of this paper is the lack of methods for estimating a correct sample size that can be used for clustering big and high dimensional datasets.”

6 Conclusiones y líneas futuras de trabajo

En este trabajo se ha desarrollado una aplicación en la nube orientada a la compartición de recursos, tanto lógicos, como físicos. Esta aplicación se ha desarrollado dentro del proyecto de I+D+i, SITAC: Social Internet of Things – Apps by and for the Crowd ITEA 2 11020, y se ha creado una plataforma para facilitar la cooperación de usuarios mediante un modelo inspirado en las redes sociales, y que ofrece un servicio web orientado al intercambio y compartición de contenido, que además ofrece recomendaciones de contenido basadas en las preferencias de los usuarios.

Como consecuencia del trabajo realizado he llegado a las siguientes conclusiones:

- Gracias a la evolución del internet de las cosas y a las tecnologías en la nube disponibles se pueden desarrollar aplicaciones web basadas en el almacenamiento y compartición de recursos orientados al big data, eficientes y con un coste ajustado, como queda probado con este trabajo.

- Durante el desarrollo del RSC he comprobado que el modelo PAAS de desarrollo y despliegue de servicios en la nube es el más adecuado a las aplicaciones big data fundamentalmente por su capacidad de elasticidad (provisión/desprovisión de recursos en función del tráfico entrante).

- Para ahorrar costes el diseño de aplicaciones big data en la nube se debe minimizar el tráfico saliente de los servidores, posibilitando el acceso a los recursos de almacenamiento directamente desde los clientes. Esto permite evitar la saturación de los servidores a la vez que reduce el coste del servicio, ya que no solo se gasta menos CPU sino menos tráfico saliente de los servidores, que es un recurso caro.

- Durante el desarrollo de la aplicación he llegado a la conclusión de que de cara a realizar un servicio web escalable es mejor implementarlo usando un lenguaje asíncrono como Node.js que Python ya que gracias a la funcionalidad del callback Node.js es más escalable al poder permitir que se puedan lanzar todas las llamadas y continuar con la ejecución sin que esta se detenga esperando las respuestas.

- Durante el desarrollo del módulo Analytics he comprobado que el uso del lenguaje de una persona media se puede clasificar en 25 grupos sin que haya problemas de falta de representación de las personas en estos grupos. Esta clasificación está relacionada con el uso que hacemos los humanos del lenguaje, modelado en base a las 300 palabras que de media se usan con más frecuencia de un total de no más de 1000. Prueba de ello además

son los trabajos analizados en el capítulo 4 donde se estudian artículos de divulgación que validan estas conclusiones.

Como líneas de trabajo futuro este proyecto creo interesante proponer las siguientes:

- Mejorar el modelo de redes sociales, aumentar la gestión de los círculos permitiendo la creación de círculos nuevos y poder relacionarse directamente con usuarios (en vez de círculos de dos personas) al igual que se hace con los amigos en otras redes sociales.

- Para hacer el servicio más agradable a nuevos usuarios podría realizarse una gestión de perfiles más compleja, y también más parecida a los modelos de redes sociales, donde se puede poner una foto del usuario o comentar una breve autobiografía cuyas palabras claves formasen a pasar parte de tu perfil.

- Una mejora que haría al servicio más competitivo sería la opción de compartir además de tus propios ficheros y carpetas, poder compartir también carpetas en Google Drive aunque para acceder a las mismas habría que tener permiso de Google.

- Otra mejora que aumentaría el valor del servicio sería la generación de un cliente de la aplicación para Windows, para Android y para Ios, al igual que ocurre con Dropbox y permitiría una inclusión de la aplicación mucho mayor.

Referencias

1 S. Datta and V. López, "El desequilibrio socioeconómico consecuente de la industria de Internet de las Cosas," *Journal Politécnico Grancolombiano Punto de Vista,* 2016.

2 V. López and e. al, "Big+Open Data: some applications for a Smartcity”," in *Proceedings of the 3th Conference on Progress in Informatics and Computing*, Nanjing, 2015.

3 P. Mell and T. Grance, "The NIST Definition of Cloud," *National Institute of Standards and Technology Special Publication 800-145,* p. 7, 2011.

4 Amazon Web Services, Inc. o sus empresas afiliadas., "Amazon Web Service," Amazon, 2016. [Online]. Available: https://aws.amazon.com/es/what-is-aws/.

5 Google, "Google cloud Platform," [Online]. Available: https://cloud.google.com/.

6 Microsoft, "Microsoft Azure," Microsoft, [Online]. Available: https://azure.microsoft.com/es-es/overview/what-is-azure/.

7 RedHat, "Openshift Online," Red Hat, [Online]. Available: https://www.openshift.com/web-hosting/index.html?sc_cid=701600000011p9xAAA&gclid=Cj0KEQjwosK4BRCYhsngx4_SybcBEiQAowaCJWbAVKCg1jcXZHqDeNkk-vHH2KWvOC6V8J8wGLgmmEIaAjbk8P8HAQ.

8 "General Python FAQ," Python Software Foundation, 7 May 2016. [Online]. Available: https://docs.python.org/2/faq/general.html#what-is-python. [Accessed 05 May 2016].

9 "History and License," Python Softare Foundation, 07 May 2016. [Online]. Available: https://docs.python.org/2/license.html.

10 T. Peters, "PEP Zen in Python," 2004. [Online]. Available: https://www.python.org/dev/peps/pep-0020/.

11 "AWS SDK for Python," Amazon Web Services, 2016. [Online]. Available: http://aws.amazon.com/es/sdk-for-python/.

12 A. Ronacher, "Flask (A Python micro framework)," 2016. [Online]. Available: http://flask.pocoo.org/docs/0.10/foreword/#what-does-micro-mean.

13 D. Flanagan, Javascript: The definitve guide, O'Reilly Media, Inc., 2006.

14 C. Severance, "JavaScript: Designing a Language in 10 Days," *IEEE Computer Society,* vol. 45, no. 02, pp. 7-8, 2012.

15 M. Andreessen and B. Eich, "INNOVATORS OF THE NET: BRENDAN EICH AND JAVASCRIPT," Netscape, 1998. [Online]. Available: https://web.archive.org/web/20080208124612/http://wp.netscape.com/comprod/columns/techvision/innovators_be.html.

16 campusMVP, "ECMAScript 6 es ya un estándar cerrado," Campus MVP, 19 Junio 2015. [Online]. Available: http://www.campusmvp.es/recursos/post/ECMAScript-6-es-ya-un-estandar-cerrado.aspx.

17 D. Flanagan, in *JavaScript: The Definitive Guide (5th ed.)*, O'Reilly & Associates, 2006, p. 16.

18 Mozilla Developer Network and individual contributors., "JavaScript data types and data structures," Mozilla, 20 March 2016. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures.

19 D. Herman, "Effective Javascript," Addison-Wesley, 2013, p. 83.

20 Amazon Web Services, Inc, "AWS SDK for JavaScript in the Browser," Amazon Web Services, Inc, 2016. [Online]. Available: https://aws.amazon.com/es/sdk-for-browser/.

21 A. Handy, "Node.js pushes JavaScript to the server-side," 24 June 2011. [Online]. Available: http://sdtimes.com/node-js-pushes-javascript-to-the-server-side/.

22 R. Dahl, "Joyent and node," Google Groups, 09 November 2010. [Online]. Available: https://groups.google.com/forum/#!topic/nodejs/lWo0MbHZ6Tc.

23 R. Dahl, "Porting Node to Windows With Microsoft's Help," Node.js Foundation, 23 June 2011. [Online]. Available: https://nodejs.org/en/blog/uncategorized/porting-node-to-windows-with-microsofts-help/.

24 P. Teixeira, Professional Node.js: Building Javascript Based Scalable Software, John Wiley and Sons.

25 Linux Foundation, "Node.js Foundation Advances Community Collaboration, Announces New Members and Ratified Technical Governance," Linux Foundation, 16 June 2015. [Online]. Available: http://www.linuxfoundation.org/news-media/announcements/2015/06/nodejs-foundation-advances-community-collaboration-announces-new.

26 Node.js Foundation, "Node.js Foundation Combines Node.js and io.js Into Single Codebase in New Release," Node.js Foundation, 14 September 2015. [Online]. Available: https://nodejs.org/en/blog/announcements/foundation-v4-announce/.

27 "Node.js w/1M concurrent connections!," [Online]. Available: http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/.

28 Node.js foundation, "About Node.js," Node.js foundation, [Online]. Available: https://nodejs.org/en/about/.

29 "Express, Infraestructura web rápida, minimalista y flexible para Node.js," Node.js, [Online]. Available: http://expressjs.com/es/.

30 "Utilización de motores de plantilla con Express," Node.js, [Online]. Available: http://expressjs.com/es/guide/using-template-engines.html.

31 Amazon Web Services, "AWS Sdk para JavaScript en Node.js," Amazon Web Services, [Online]. Available: https://aws.amazon.com/es/sdk-for-node-js/.

32 C. Ullman, "What is Ajax?," in *Beginning Ajax*, Wiley Publishing Inc, 2007, p. 498.

33 J. J. Garret, "Ajax: A New Approach to Web Applications," Adaptative Path, 18 February 2005. [Online]. Available: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/.

34 "Remote scripting with javascript," dotvoid, 13 August 2002. [Online]. Available: http://web.archive.org/web/20080403223512/http://www.dotvoid.com/view.php?id=13.

35 D. Crockford, "Douglas Crockford: The JSON Saga," [Online]. Available: https://www.youtube.com/watch?v=-C-JoyNuQJs.

36 Ecma International, "Final draft of the TC39 “The JSON Data Interchange Format” standard," 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

37 M. Otto, "Bootstrap from Twitter," Twitter, Inc, 19 August 2011. [Online]. Available: https://blog.twitter.com/2011/bootstrap-from-twitter.

38 M. Otto, "Bootstrap 3.3.0 released," 29 October 2014. [Online]. Available: http://blog.getbootstrap.com/2014/10/29/bootstrap-3-3-0-released/.

39 "Apache JMeter," Apache Software Foundation, [Online]. Available: http://jmeter.apache.org/.

40 R. Barranco Fragoso, "¿Qué es Big Data?," 18 June 2012. [Online]. Available: https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/.

41 A. Martín, S. Chavez, N. Rodríguez, A. Valenzuela and M. Murazzo, "Bases de datos NoSql en cloud computing," April 2013. [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/27121/Bases+de+Datos+NoSql+en+Cloud+Comp

uting.pdf?sequence=1.

42 R. Hecht and S. Jablonsky, "NoSQL Evaluation a use case oriented survey," in *International Conference on Cloud and Service Computing.*, 2011.

43 M. B. Bianchi Widder, "Els beneficis de l'ús de tecnologies NOSQL," Facultat d'Informàtica de Barcelona, Barcelona, 2012.

44 A. K. Pujari, Data Mining Techinques, Universities Press, 2001.

45 S. Chakrabarti, M. Ester, U. Fayyad, J. Gehrke, J. Han, S. Morishita, G. Piatetsky-Shapiro and W. Wang, "Data Mining Curriculum: A Proposal," 2006.

46 M. Ester, H.-P. Kriegel, J. Sander and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96),* pp. 226-231, 1996.

47 O. Maimon and L. Rokach, "Clustering methods," in *Data mining and knowledge discovery handbook.*, Springer, 2005, pp. 321-352.

48 R. Sibson, "SLINK: an optimally efficient algorithm for the single-link cluster method," *The Computer Journal,* pp. 30-34, 1973.

49 "An efficient algorithm for a complete link method," *The Computer Journal,* pp. 364-366, 1977.

50 M. Mahajan, P. Nimbhorkar and K. Varadarajan, "The Planar k-Means Problem is NP-Hard," in *WALCOM: Algorithms and Computation*, vol. 5431, Springer Berlin Heidelberg, 2009, pp. 274-285.

51 "Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering," in *SCG '94 Proceedings of the tenth annual symposium on Computational geometry*, New York, ACM, 1994, pp. 332-339.

52 D. MacKay, "Chapter 20. An Example Inference Task: Clustering," in *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, 2003, p. 284–292.

53 D. Martin, "How can we choose a "good" K for K-means clustering?," Quora, 06 December 2013. [Online]. Available: https://www.quora.com/How-can-we-choose-a-good-K-for-K-means-clustering.

54 J. B. Lovins, "Development of a Stemming Algorithm," *Mechanical Translation and Computational Linguistics,* vol. 11, no. 1, 2, pp. 22,31, 1968.

55 M. Porter, "PorterStemmer," [Online]. Available: http://tartarus.org/~martin/PorterStemmer/index.html.

56 M. Porter, "Snowball," [Online]. Available: http://snowball.tartarus.org/.

57 M. Porter, "Spanish stemming algorithm," [Online]. Available: http://snowball.tartarus.org/algorithms/spanish/stemmer.html.

58 O. Foundation, "Open ID Connect," OpenID Foundation, [Online]. Available: http://openid.net/connect/.

59 D. Hardt, "RFC 6749 The OAuth 2.0 Authorization Framework," Microsoft, October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749.

60 O. C. Workgroup, "OAuth Core 1.0," 4 December 2007. [Online]. Available: http://oauth.net/core/1.0/.

61 A. Kerckhoffs, "La cryptographie militair," *Journal des sciences militaires,* vol. IX, pp. 5-83, 1883.

62 P. Ragone, "A spanish stemmer / Un lematizador de español," [Online]. Available: http://stemmer-es.sourceforge.net/.

63 K. Mitofsky. [Online]. Available: https://jsfiddle.net/KyleMit/X9tgY/. [Accessed 2015].

64 L. Page, "Method for node ranking in a linked database". Patent US6285999 B1, 4 September 2001.

65 M. Albert, 2014. [Online]. Available: https://pypi.python.org/pypi/cluster/1.2.2.

66 M. Davies, "Vocabulary Range and Text Coverage:Insights from the Forthcoming Routledge Frequency Dictionary of Spanish," *Selected Proceedings of the 7th Hispanic Linguistics Symposium,* pp. 106-115, 2005.

67 A. Arguelles, March 2005. [Online]. Available: http://how-to-learn-any-language.com/forum/forum_posts.asp?TID=267&PN=0&TPN=1.

68 "http://factoides.com.ar/post/3031124497/usamos-solo-unas-300-palabras," [Online].

69 J. J. Garcia Aranda and J. Ramos Diaz, "HD-Python-Cluster," [Online]. Available: https://github.com/jjaranda13/HD-Python-cluster.

70 "sklearn.cluster.KMeans," [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

71 "Sort Journal," [Online]. Available: http://www.idescat.cat/sort/.

Anexo.

Anexo 1: Herramientas

Para la realización de este trabajo se han usado las siguientes herramientas:

- Amazon Web Services y sus Apis y servicios
    - Boto para Python y AWS SDK for node para node
    - AWS SDK for javascript para el desarrollo de las funciones de AWS en JavaScript.
- Frameworks para programar los servidores
    - Flask para Python
    - Express para Node.js
- Twitter Bootstrap
- Apache JMeter
- Herramientas de programación
    - Sublime text para programar la parte cliente del servicio, y la parte servidor en Node.js
    - Python IDLE para el desarrollo en python
- El navegador usado para probar el servicio ha sido Google Chrome, excepto para realizar consultas sobre la base de datos SimpleDB que se ha usado Mozilla Firefox ya que posee la extensión SDB Tool.
- Microsoft Word 2007 para la realización de la memoria y toda la documentación asociada al proyecto.

Anexo 2: Mejoras librería de cluster

En esta sección se detallan los cambios realizados en la librería cluster 1.2.2 para adaptarla a este proyecto.

Reducción de la complejidad

En este apartado vemos los cambios realizados en la función asigna_item() de la implementación del método K-Means.

En el original se recalculan los centroides cada vez que se reasigna un ítem a un cluster, y esto no es siempre necesario, en la nueva implementación este cálculo se ahorra ya que se lleva calculada una estructura de tipo diccionario en la que se almacenan los cluster asociados al centroide que los representa.

Versión original:

```
def assign_item(self, item, origin):
        """
        Assigns an item from a given cluster to the closest located cluster.

        :param item: the item to be moved.
        :param origin: the originating cluster.
        """
        closest_cluster = origin
        for cluster in self.__clusters:
            if self.distance(item, centroid(cluster)) < self.distance(
                    item, centroid(closest_cluster)):
                closest_cluster = cluster

        if id(closest_cluster) != id(origin):
            self.move_item(item, origin, closest_cluster)
            return True
        else:
            return False
```

Versión actualizada:

```
def HDassign_item(self, item, origin, origin_centroid, my_centroids):
        """
        Assigns an item from a given cluster to the closest located cluster.

        :param item: the item to be moved.
        :param origin: the originating cluster.
        :param origin_centroid: centroid of the originating cluster
        :my_centroids: dictionary of centroid,cluster
        """
        closest_cluster=origin
#my_centroids[closest_centroid]=closest_cluster
        closest_centroid=origin_centroid
```

```
        #for cluster in self.__clusters:
        for centro in my_centroids.keys():
            if self.distance(item, centro) < self.distance(
                    item, closest_centroid):
                closest_cluster = my_centroids[centro]

        if id(closest_cluster) != id(origin):
            self.move_item(item, origin, closest_cluster)
            return True
        else:
            return False
```

El cálculo de my_centroids se hace en la función HDgetClusters():

Optimización del tiempo de computación

En esta sección se muestra como se limita el número de iteraciones de la función getClusters(), para que cualquier cambio mínimo que realmente no afecta al resultado no implique un nuevo cálculo de todos los clusters.

Función original:

```
def getclusters(self, count):
        """
        Generates *count* clusters.

        :param count: The amount of clusters that should be generated.  count
            must be greater than ``1``.
        :raises ClusteringError: if *count* is out of bounds.
        """

        # only proceed if we got sensible input
        if count <= 1:
            raise ClusteringError("When clustering, you need to ask for at "
                                  "least two clusters! "
                                  "You asked for %d" % count)

        # return the data straight away if there is nothing to cluster
        if (self.__data == [] or len(self.__data) == 1 or
                count == self.__initial_length):
            return self.__data

        # It makes no sense to ask for more clusters than data-items
available
        if count > self.__initial_length:
            raise ClusteringError(
                "Unable to generate more clusters than "
                "items available. You supplied %d items, and asked for "
                "%d clusters." % (self.__initial_length, count))

        self.initialise_clusters(self.__data, count)
```

```
        items_moved = True  # tells us if any item moved between the
clusters,
                            # as we initialised the clusters, we assume that
                            # is the case

        while items_moved is True:
            items_moved = False
            for cluster in self.__clusters:
                for item in cluster:
                    res = self.assign_item(item, cluster)
                    if items_moved is False:
                        items_moved = res
        return self.__clusters
```

Función actualizada:

```
def HDgetclusters(self, count,  max_iterations):
        """
        Generates *count* clusters.

        :param count: The amount of clusters that should be generated.  count
            must be greater than ``1``.
        :raises ClusteringError: if *count* is out of bounds.
        """

        # only proceed if we got sensible input
        if count <= 1:
            raise ClusteringError("When clustering, you need to ask for at "
                                  "least two clusters! "
                                  "You asked for %d" % count)

        # return the data straight away if there is nothing to cluster
        if (self.__data == [] or len(self.__data) == 1 or
                count == self.__initial_length):
            return self.__data

        # It makes no sense to ask for more clusters than data-items
available
        if count > self.__initial_length:
            raise ClusteringError(
                "Unable to generate more clusters than "
                "items available. You supplied %d items, and asked for "
                "%d clusters." % (self.__initial_length, count))

        self.initialise_clusters(self.__data, count)

        items_moved = True  # tells us if any item moved between the
clusters,
                            # as we initialised the clusters, we assume that
                            # is the case

        iteration=0
```

```
        #asi no, no obligar a hacer iteraciones, lo hago segun dice el
algoritmo
        #pero si llego a iteraciones paro, si termino antes de llegar, mejor
        while items_moved is True:
            items_moved = False
            print "iterating",iteration
            ts = time.time()
            st=datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d
%H:%M:%S')
            print st
            iteration=iteration+1

            #computation of centroids
            my_centroids={}    # new!!
            for cluster in self.__clusters:# new!!
                one_centroid=HDcentroid(cluster)# new!!
                my_centroids[one_centroid]=cluster # new!!


            #now we scan the N items without recalculation of centroids.
Therefore, it is linear
            for cluster in self.__clusters:
                for centroid_aux, cluster_aux in my_centroids.iteritems():
                    if cluster_aux == cluster:
                        centroid_cluster=centroid_aux
                        break;
                for item in cluster:
                    res = self.HDassign_item(item,
cluster,centroid_cluster,my_centroids)#modified!!
                    if items_moved is False:
                        items_moved = res

            if (iteration == max_iterations):
                items_moved = False
        return self.__clusters
```

Datos específicos del RSC y alta dimensionalidad

Para adaptar los datos del RSC a la librería de clustering se ha definido una función distancia específica:

```
def HDdistItems(profile1,profile2):
    #Distance function, this distance between two profiles is based on:
    #For each keyword of user A, if the keyword is not  present in user B ,
then the distance for this keyword is the weight in the user A.
    #If the keyword exists in both users, the weights are compared and the
distance is the absolute difference
    len1=len(profile1)/2
    len2=len(profile2)/2
    total_len=len1+len2 #this value usually is 20
    factor_len=20/total_len #this only work if the profile has less than 10
keys
    distance = 0.0
    marked=[0]*20;
```

```
    for i in range(len1):
        found=False
        for j in range(len2):
            if profile1[i*2]==profile2[j*2]:
                distance+=abs(profile1[i*2+1]-profile2[j*2+1]);
                found=True;
                marked[j*2]=1;
                break;
        if found==False:
            distance+=profile1[i*2+1];

    for i in range(len2):
        if marked[i*2]==1:
            continue;
        distance+=profile2[i*2+1]

    distance=distance*factor_len
    return distance
```

Y funciones para el cálculo de la métrica SSE:

```
def HDcomputeSSE(solution,numclusters):
    #This metric measure the cohesion of users into a cluster and the
separation among clusters at the same time
    partial_solution=0
    total_solution=0
    dist=0
    for i in range(numclusters):
        partial_solution=0
        for j in solution[i]:
            dist=HDdistItems(util.HDcentroid(solution[i]),j)
            partial_solution+=dist*dist
            total_solution+=partial_solution
    return total_solution


def HDcomputeSSEfromCentroids(items,centroids):
    #This metric measure the cohesion of users into a cluster and the
separation among clusters at the same time
    sse=0
    dist=0
    print
    for i in items:

    #for each item determine the best centroid
        dist_centroid=100000
        for j in centroids:
            dist_centroid_aux=HDdistItems(i,j)
            if dist_centroid_aux< dist_centroid:
                centroid=j
                dist_centroid=dist_centroid_aux
        # dist_centroid is the dist from user to its assigned centroid (which
is j)
        #print "d:", dist_centroid
        sse+=dist_centroid*dist_centroid
    return sse
```

Además se ha cambiado el cálculo del centroide, el original era:

```
def centroid(data, method=median):
    "returns the central vector of a list of vectors"
    out = []
    for i in range(len(data[0])):
        out.append(method([x[i] for x in data]))
    return tuple(out)
```

Y la nueva versión es:

```
def HDcentroid(data):
    dict_words={}
    dict_weight={}
    words_per_user=10 #10 words per user. This value is not used.
    num_users_cluster=len(data)# len(data) is the number of users (user=item)

    for i in range (num_users_cluster):
        words_per_user=len(data[i])/2 #each profile have 10 pairs of keyword,
weight
        for j in range (words_per_user):
            word=(data[i])[j*2]
            if (dict_words.has_key(word)) :
                dict_words[word]+=1
                dict_weight[word]+=data[i][2*j+1]
            else :
                dict_words[word]=1
                dict_weight[word]=data[i][2*j+1]
    #l is a ordered list of the keywords, with the sum of the weight of every
popular keyword
    l=dict_words.items()
    l.sort(key=lambda x:10000000-x[1])

    words_per_centroid=min(10,len(l))

    out=[0]*words_per_centroid*2
    centroid_total_weight=0

    for i in range  (words_per_centroid):
        tupla=l[i] # word, sum of weights
        out[i*2]=tupla[0]
        out[i*2+1]=dict_weight[tupla[0]]/tupla[1]
        centroid_total_weight+=out[i*2+1]
    #normalization of the centroid weight
    for i in range(words_per_centroid):
        out [i*2+1]=out[i*2+1]/centroid_total_weight
    return tuple(out)
```

Anexo 3: Integración de la autenticación delegada.

En esta sección se explicará el proceso necesario para integrar el método de autenticación delegada en el servicio:

Para completar la autenticación delegada hay que implementar el botón de acceso dentro de la aplicación, que entregará un token, con el que se podrá entrar definitivamente.

En el caso del RSC este botón formará parte de un formulario que contendrá los datos de autenticación con Google, que al submitir, llamará a la gestión de acceso del servidor con el token de Google.

```
<form action="/hello_google" id="googlesignin" method="POST" class="form-
signin" role="form" style="display:none">
         <button class="btn btn-lg btn-danger btn-block" type="submit">Google
Sign</button>
         <input id="guser" name="guser"></input>
         <input id="gpass" name="gpass"></input>
         <input id="id_token" name="id_token"></input>
      </form>
```

Luego se incluyen las funciones que se ejecutan al cargar la página. La primera es la identificación de la aplicación en Google, para que Google pueda responderte con el token de acceso.

```
  var startApp = function() {
    gapi.load('auth2', function(){
      // Retrieve the singleton for the GoogleAuth library and set up the
client.
      auth2 = gapi.auth2.init({
        client_id: '',
        cookiepolicy: 'single_host_origin',
        // Request scopes in addition to 'profile' and 'email'
        //scope: 'additional_scope'
      });
      signOut();
      attachSignin(document.getElementById('botongoogle'));
    });
  };

  function attachSignin(element) {
    console.log(element.id);
    auth2.attachClickHandler(element, {},
        function(googleUser) {
          onSuccess(googleUser);
        }, function(error) {
          alert(JSON.stringify(error, undefined, 2));
        });
  }
```

Y luego se añaden las funciones que gestionan el funcionamiento del botón de autenticación con Google.

```
function onSuccess(googleUser) {
        console.log('Logged in as: ' +
googleUser.getBasicProfile().getName());
        var profile = googleUser.getBasicProfile();
        console.log("ID: " + profile.getId()); // Don't send this directly to
your server!
        console.log("Name: " + profile.getName());
        console.log("Image URL: " + profile.getImageUrl());
        console.log("Email: " + profile.getEmail());

        // The ID token you need to pass to your backend:
        var id_token = googleUser.getAuthResponse().id_token;
        console.log("ID Token: " + id_token);
        document.getElementById("guser").value=profile.getEmail();
        document.getElementById("gpass").value="1234";
        document.getElementById("id_token").value=id_token;
        document.getElementById("googlesignin").submit();
        //logueado=true;
      }
function onFailure(error) {
        console.log(error);
      }
function renderButton() {

      gapi.signin2.render('my-signin2', {
          'scope': 'https://www.googleapis.com/auth/plus.login',
          'width': 200,
          'height': 50,
          'longtitle': true,
          'theme': 'dark',
          'onsuccess': onSuccess,
          'onfailure': onFailure
        });
      }


function signOut() {
    var auth2 = gapi.auth2.getAuthInstance();
    auth2.signOut().then(function () {
      console.log('User signed out.');
    });}
```

En la Figura 88 se muestra el botón integrado en la aplicación:

Figura 86: Página de login con acceso mediante tu cuenta de Google

Anexo 4: Pruebas de carga con Apache JMeter

En este apartado se describen las pruebas de carga realizadas con la herramienta Apache JMeter, para establecer el valor en el que el disparador de elasticidad de Elastic Beanstalk empezará a actuar y a crear nuevas instancias de la máquina en la que esta cargado el código.

1 Peticiones

En este apartado se describen las operaciones que se usarán para hacer las pruebas de carga.

1.1 Una operación de lectura

Lectura de una fila de la tabla USER_PROFILE_TABLE.

URL: http://mms-env-node.elasticbeanstalk.com/nodemodule/getProfile

Método: POST

Atributos: user_id=user_id_demo, top_n= 10

1.2 N operaciones de lectura 1 operación de escritura

Abrir cursor en MMS_RESOURCE_TABLE leer N filas, reconstruir el perfil y escribirlo en USER_PROFILE_TABLE

URL: http://mms-env-node.elasticbeanstalk.com /nodemodule/getRebuildProfile

Método: POST

Atributos: user_id=user_id_demo

Este es el escenario más interesante porque involucra búsquedas en una tabla de N filas y una escritura. Que es parecido al caso de uso más común del RSC en el que un usuario realiza una búsqueda, en la que se leen N filas, y se escoge una de ellas para incorporarla al inventario de terceros.

1.3 Una Escritura

URL: http://mms-env-node.elasticbeanstalk.com/nodemodule/createService

user_id=user_id_demo

service_url= http://blabla.com

service_name= prueba

functional_params=[]

non_functional_params=[]

1.4 Un Borrado

http://mms-env-node.elasticbeanstalk.com/nodemodule/deleteService

user_id=user_id_demo

service_url= http://blabla.com

service_name:”prueba”

2. Estrategia para los tests

Tipos de instancia: single or autoscaling

Inicialmente para medir el rendimiento de una máquina se hacen las pruebas con una *single* instance, una vez se alcance el límite podemos definir el valor umbral del disparador de autoescalado de Elastic Beanstalk.

3. Parámetros de auto escalado

Parámetros generales:

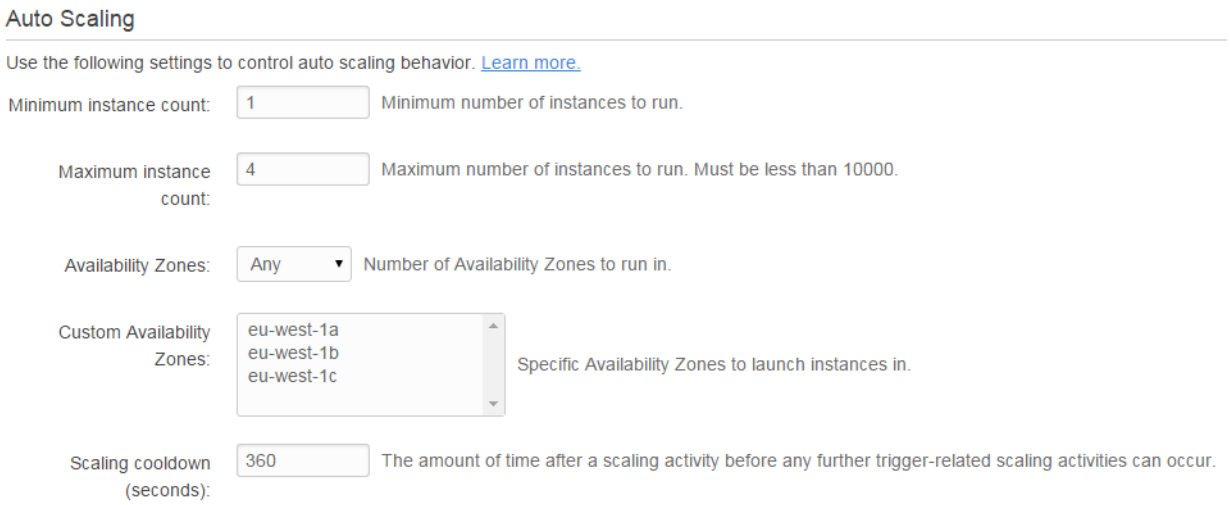

Figura 87: Parámetros generales de auto escalado

Para hacer las mediciones nos basaremos en la medida de la latencia, que nos indicará el consumo de CPU de la aplicación. Esto se ve en la Figura 90.

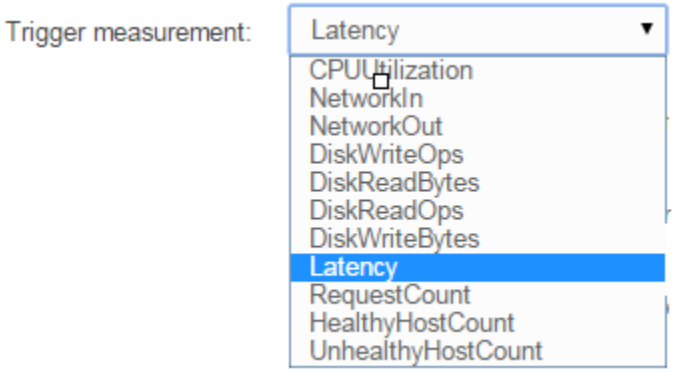

Figura 88: Medida para el uso del disparador de autoescalado

El resto de parámetros se muestran en la Figura 91.

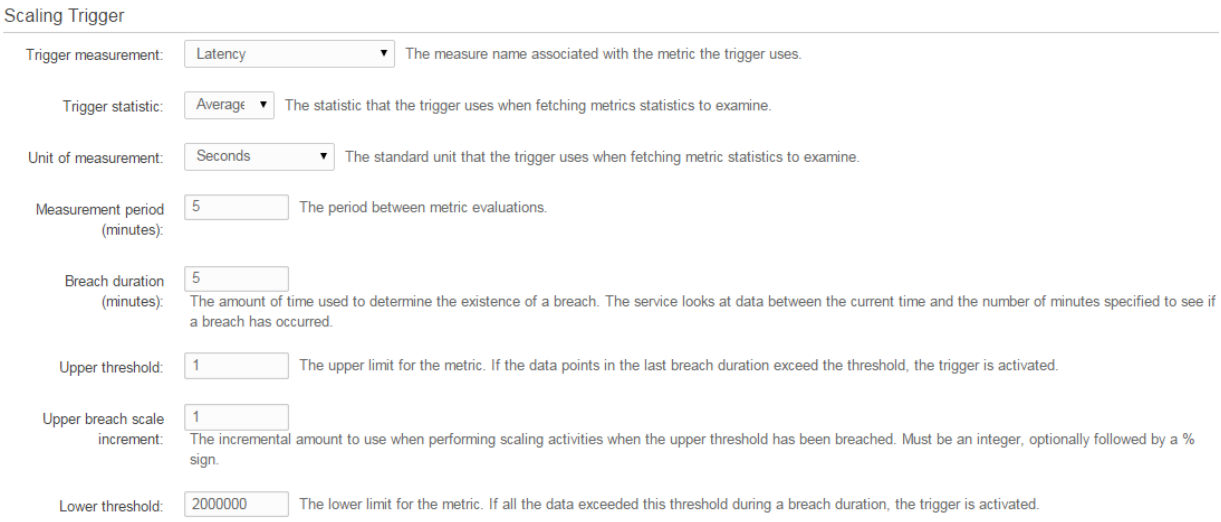

Figura 89: Resto de parámetros disparador de autoescalado

4. Escenarios y resultados

En esta sección se describen los escenarios de las pruebas y sus resultados, basándose en:

- Las pruebas se realizan con la herramienta Apache JMeter.
- Las medidas en JMeter y Amazon son diferente puesto que JMeter añade latencia de red, que en nuestro caso son 165ms segun se calcula en el primer escenario.
- La máquina EC2 es la micro.t1
- La configuración de JMeter para los tests será:
    - *Thread group*: 1000 usuarios
    - *Ramp up*: 10 segundos
    - Test durante 30 segundos.

4.1 Resultados de una lectura simple

En este apartado se analizan las pruebas de carga para distintos números de usuario.

Carga de un usuario

Jmeter: media= 275ms, mediana=280

El RTT puede ser calculado usando la mediana. De todas formas la variación en el cálculo es errática, en las repeticiones del experimento usando la mediana, el RTT es constante. En el caso de este escenario particular, la mediana y la media tienen valores muy similares pero en otros

esto no ocurre porque JMeter aumenta el rendimiento en cada paso hasta alcanzar la concurrencia deseada, por lo tanto tarda algunos segundos hasta que alcanza la estabilidad.

En el grafo en JMeter de la Figura 92 la línea azul muestra el tiempo medio de respuesta, y la linea purpura muestra la mediana. Otro valor importante es el rendimiento (*throughput*) que indica el numero de peticiones procesadas por minute.

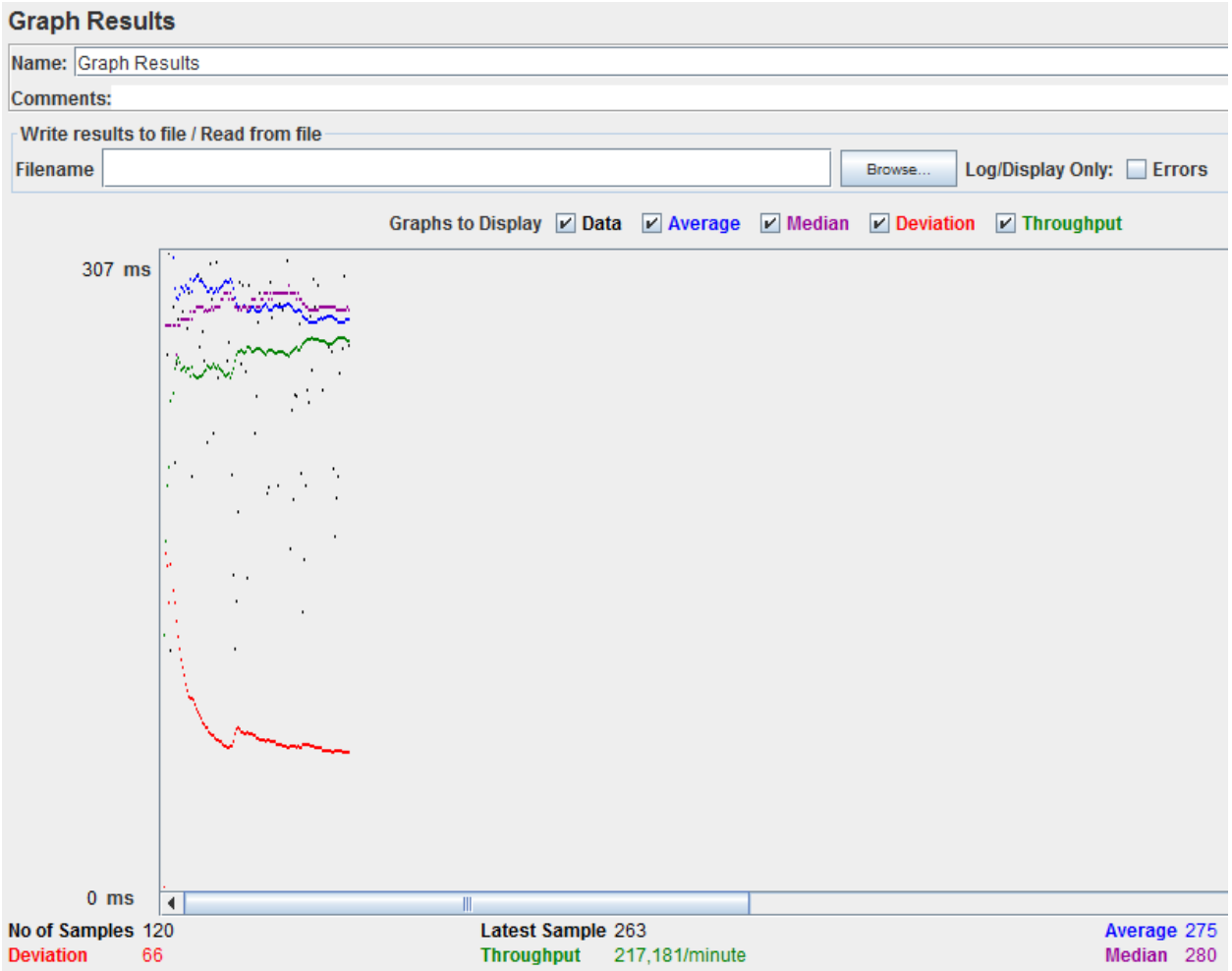

Figura 90: Resultados prueba 1

Por otro lado, cómo vemos en la Figura 93, Elastic Beanstalk ofrece un tiempo medio de respuesta de 38ms.

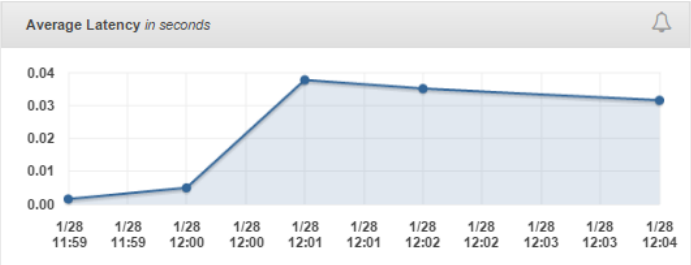

Figura 91: Tiempo medio de respuesta de Elastic Beanstalk

Conclusión: Nuestra latencia de red (RTT) es 280-38=240ms, puero puede variar en función del momento en que se ejecuta la prueba.

Carga de 10 usuarios

Jmeter: media=212ms, mediana=208ms, rendimiento=2442 peticiones/min

Medición de Amazon: 42ms

208-42= 170 (RTT).

Conclusión: No se degrada el servicio con10 usuarios concurrentes-

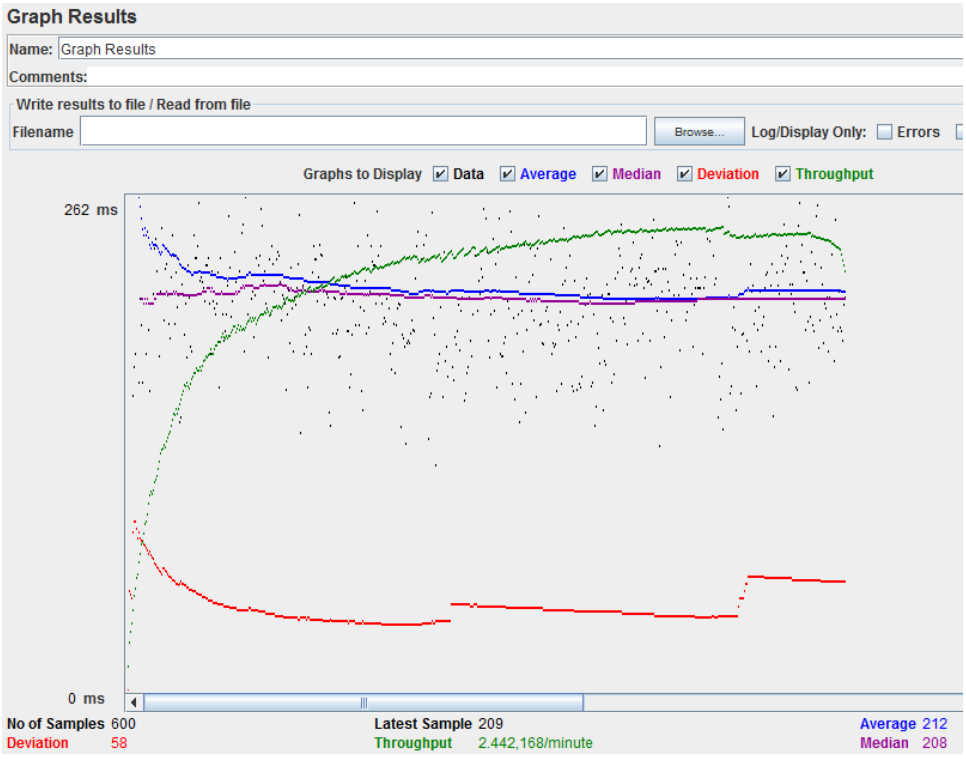

Figura 92: Prueba de carga de 10 usuarios

Carga de 50 usuarios

Medición en Amazon Elastic Beanstalk: 82 ms

Jmeter: Media=263.3ms, mediana=236, rendimiento: 4459 peticiones/min

RTT : 236-82=154

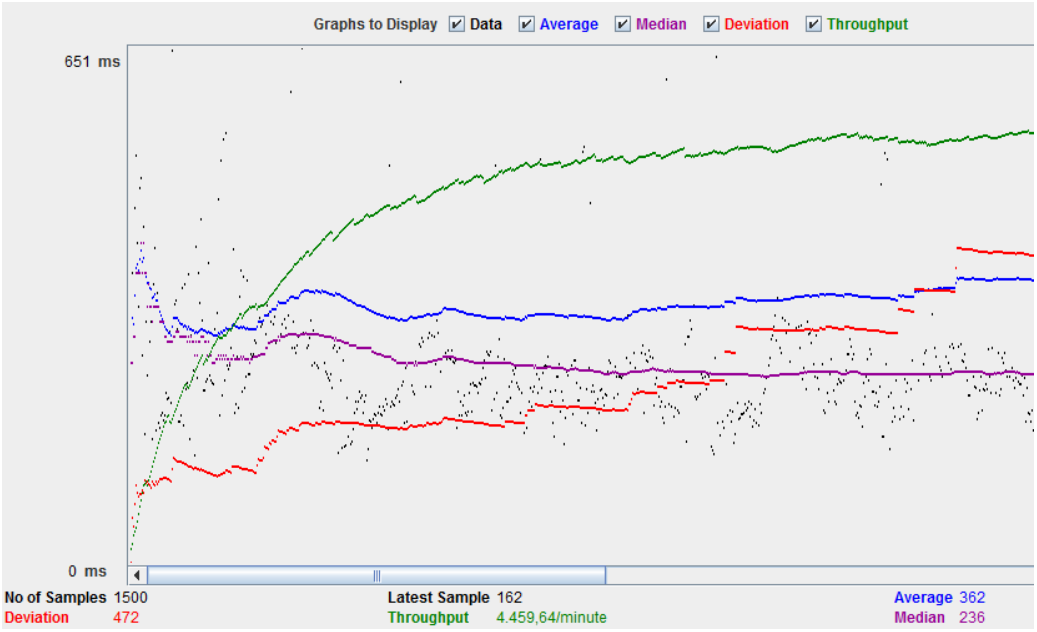

Figura 93: Prueba de carga de 50 Usuarios

Conclusión: Elastic Beanstalk empieza a degradar con la métrica de la media.

Carga de 100 usuarios

Jmeter: Mediana=263, rendimiento 3323 peticiones/min

Medición en Amazon Elastic Beanstalk: 97.4ms

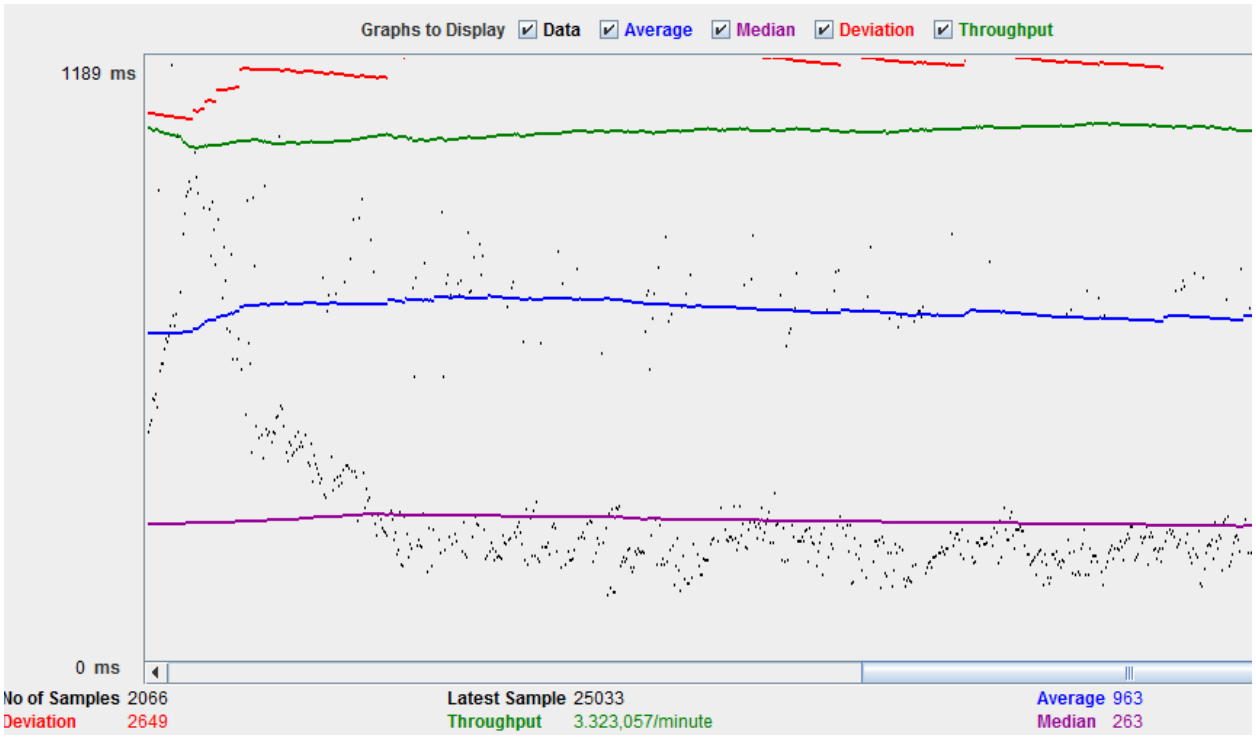

Figura 94: Prueba de carga de 100 Usuarios

Carga de 1000 usuarios

Jmeter: Rendimiento 5800peticiones/min, mediana: 4.2 seconds

La degradación es clara.

Medición en Amazon Elastic Beanstalk : 2.3 segundos

Estas métricas no son coherentes, estamos en situación de saturación.

Parece que más allá de las 5000 peticiones/min la máquina está saturada y el orden de magnitud de la latencia se incrementa (desde 263ms hasta 4 segundos). Esta barrera no puede ser cruzada.

Concurrencia en operaciones de lectura: 5000 peticiones/min como máximo.

Disparador de elasticidad para las operaciones de lectura: 1 segundo puede ser adecuado, porque normalmente la máquina responde en menos de 250ms, pero para proveer de un margen mayor se establecerá en 2 segundos que también es adecuado.

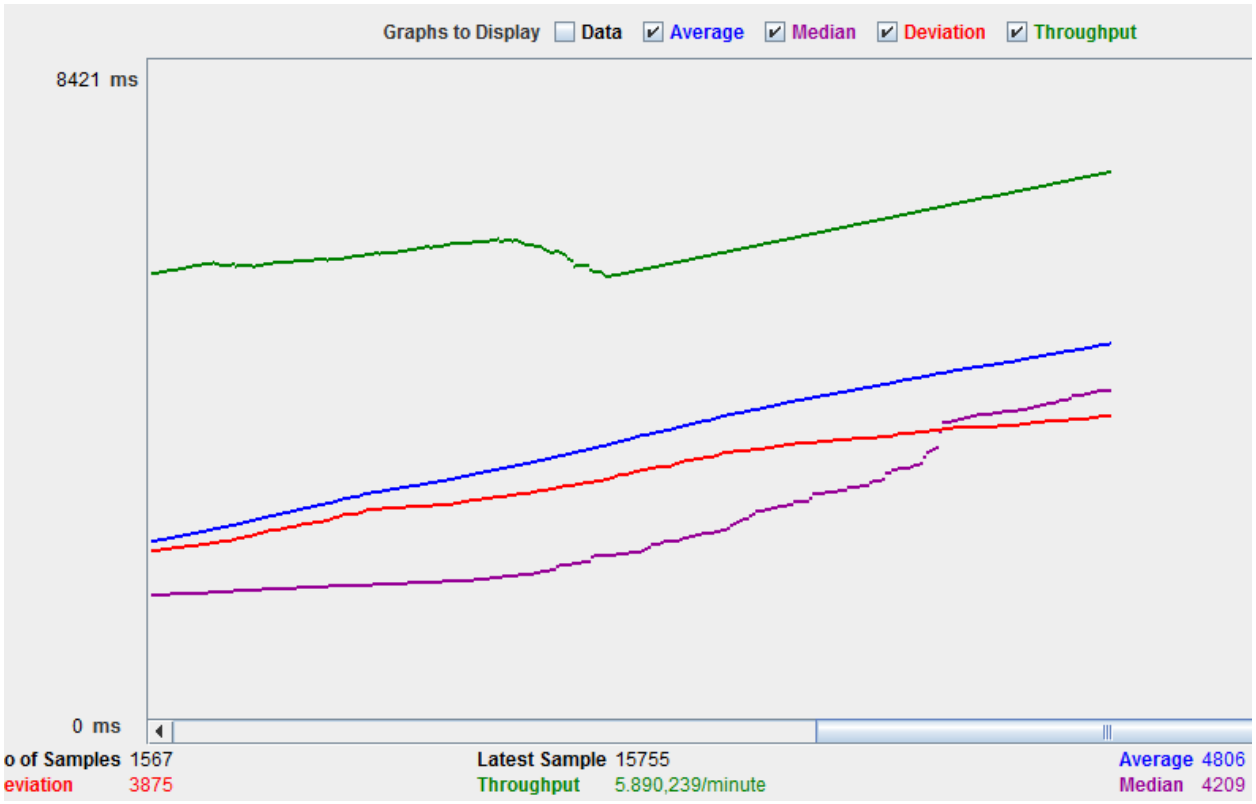

Figura 95: Prueba de carga de 1000 usuarios

4.2 Resultados de N lecturas y 1 escritura

En este apartado se muestran las pruebas para N operaciones de lectura y 1 de escritura.

Carga de 10 usuarios

Jmeter: latencia =140ms, Rendimiento:3000 peticiones/min

Medición en Amazon Elastic Beanstalk: 35.3 ms

Figura 96: Prueba de carga de 1 usuario, N lecturas 1 escritura

Carga de 100 usuarios

Jmeter: latencia =280, rendimiento=5100 peticiones/min

Medición en Amazon Elastic Beanstalk: 184ms

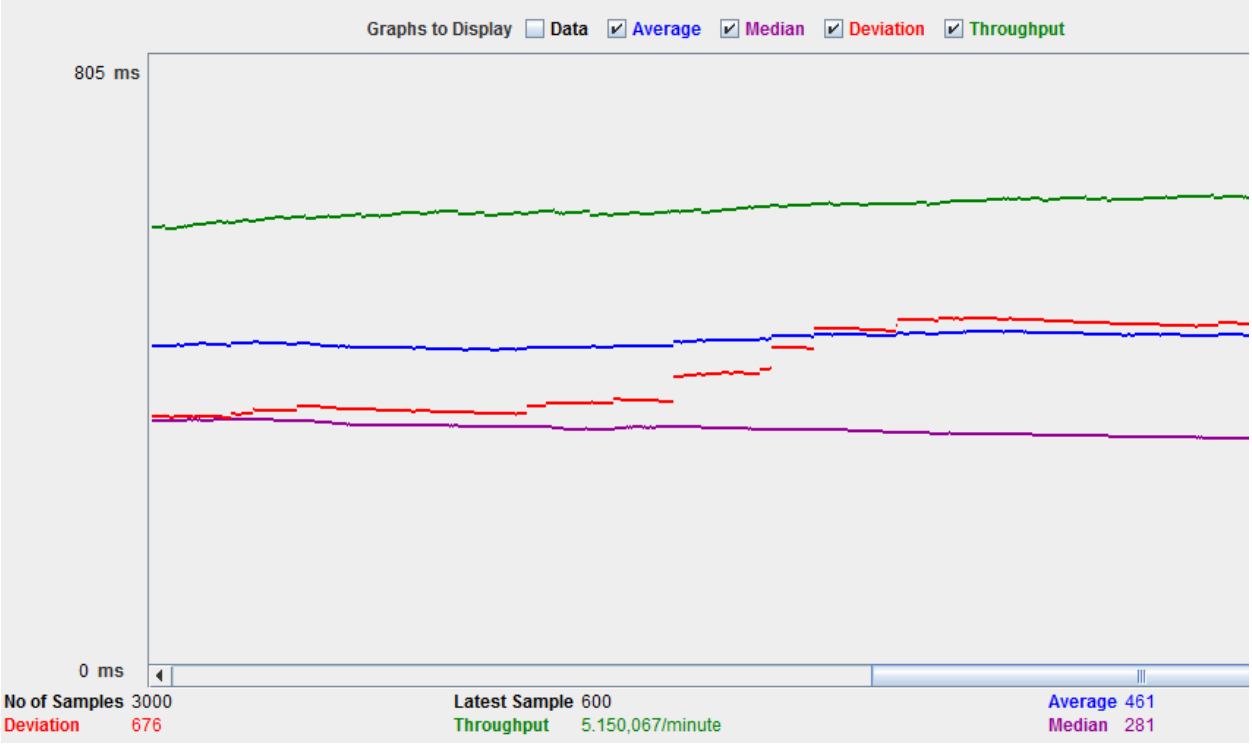

Figura 97: Prueba de carga de 100 usuarios, N lecturas 1 escritura

Carga de 200 usuarios

Jmeter: latencia = llega a alcanzar los 7 segundos, rendimiento= alcanza la 5000 peticiones/min y se decrementa hast alas 1800 peticiones/min

Medición en Amazon Elastic Beanstalk: 5 segundos

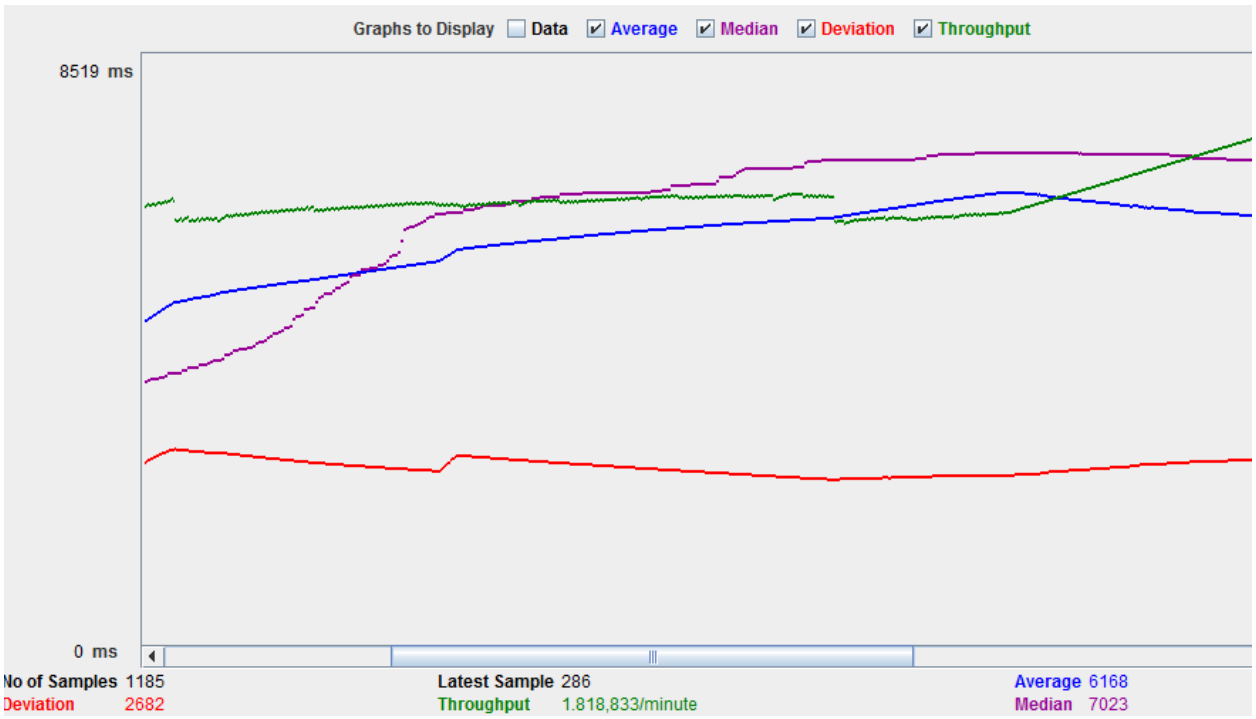

Figura 98: Prueba de carga de 200 usuarios, N lecturas 1 escritura

Anexo 5:Descripción de código

En esta sección se describe y muestra el código de algunas de las funciones más importantes.

Lematizador

El algoritmo lematizador descrito en el fichero stemmer.js es el siguiente:

```
function is_vowel(c) {
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' || c ==
'á' || c == 'é' ||
      c == 'í' || c == 'ó' || c == 'ú');
  }

  var start = 0;
  function getNextVowelPos(word, start) {
    len = word.length;//strlen(word);
    for (i = start; i < len; i++){
      if (is_vowel(word.charAt(i))) {return i;}
    }
    return len;
  }

  var start = 0;
  function getNextConsonantPos(word, start) {
    len = word.length;//strlen($word);
    for (i = start; i < len; i++){
      if (!is_vowel(word.charAt(i))) {return i;}
    }
    return len;
  }

  function endsin(word, suffix) {
    if (word.length < suffix.length) return false;
    return (word.substr(word.length-suffix.length,word.length)==suffix);
  }

  function endsinArr(word, suffixes) {
    for (i=0;i<suffixes.length;i++){
      suff=suffixes[i];
      if(endsin(word,suff)) {return suff;}
    }
    return '';
  }

  function removeAccent(word) {
    especiales = new Array('á','é','í','ó','ú');
    normales = new Array('a','e','i','o','u');
    var i = 0;
    var j = 0;
    while(i<word.length){
      for (j=0;j<especiales.length;j++){
```

```
        word = word.replace(especiales[j],normales[j]);
      }
      i++;
    }

    return word
  }

  function stemm(word) {
    var r1 = 0; var r2 = 0; var rv = 0;
    var debug = document.getElementById('intermedio');

    var len = word.length;
    if (len <= 2) {return word;}

    word = word.toLowerCase();

    r1 = r2 = rv = len;
    //R1 es la region despues de la primera consonante que sigue a una vocal,
o es la region nula al final de la palabra si no existe esa consonante.
    for (i = 0; ((i < (len-1)) && (r1 == len)); i++) {
      if ((is_vowel(word.charAt(i))) && (!(is_vowel(word.charAt(i+1))))) {
          r1 = i+2;
      }
    }

    //R2 es la region despues de la primera consonante que sigue a una vocal
en R1, o es la region nula al final de la palabra si no existe esa
conosonante.
    for (i = r1; ((i < (len -1)) && (r2 == len)); i++) {
      if ((is_vowel(word.charAt(i))) && (!(is_vowel(word.charAt(i+1))))) {
        r2 = i+2;
      }
    }

    if (len > 3) {
      if(!(is_vowel(word.charAt(1)))) {
        // Si la segunda letra es una consonante, RV es la region despues de
la siguiente vocal.
        rv = getNextVowelPos(word, 2) +1;
      } else if ((is_vowel(word.charAt(0))) && (is_vowel(word.charAt(1)))) {
        // O si las primeras dos letras son vocales, RV es la region despues
de la siguiente consonante.
        rv = getNextConsonantPos(word, 2) + 1;
      } else {
        //En otro caso (el caso consonante-vocal), RV es la region despues de
la tercera letra. Pero RV es el final de la palabra si no puedes encontrar
esas posiciones.
        rv = 3;
      }
    }

    //Ahora mismo estoy suponiendo que r1, r2 y rv nunca son negativos (si lo
fuesen cogerian el substr desde el final de la cadena).
    var r1_txt = word.substr(r1, word.length); //$r1_txt = substr($word,$r1);
    var r2_txt = word.substr(r2, word.length); //$r2_txt = substr($word,$r2);
    var rv_txt = word.substr(rv, word.length); //$rv_txt = substr($word,$rv);
```

```
    var word_orig = word;

    //Paso 0: Pronombre adjunto. // Step 0: Attached pronoun
    var pronoun_suf = new Array('me', 'se', 'sela', 'selo', 'selas', 'selos',
'la', 'le', 'lo', 'las', 'les', 'los', 'nos');
    var pronoun_suf_pre1 = new Array('éndo', 'ándo', 'ár', 'ér', 'ír');
    var pronoun_suf_pre2 = new Array('ando', 'iendo', 'ar', 'er', 'ir');
    var suf = '';
    var pre_suff = '';
    suf = endsinArr(word, pronoun_suf);

    if (suf != '') {
      pre_suff = endsinArr(rv_txt.substr(0,(rv_txt.length-
suf.length)),pronoun_suf_pre1);
      if (pre_suff != '') {
        word = removeAccent(word.substr(0,(word.length-suf.length)));
      } else {
        pre_suff = endsinArr(rv_txt.substr(0,(rv_txt.length-
suf.length)),pronoun_suf_pre2);
        if ((pre_suff != '') ||
          ((endsin(word, 'yendo' )) &&
          (word.substr(((word.length-suf.length)-6),(((word.length-
suf.length)-6)+1)) == 'u'))) {
          word = word.substr(0,(word.length-suf.length));
        }
      }
    }

    if (word != word_orig) {
      r1_txt = word.substr(r1,word.length);
      r2_txt = word.substr(r2,word.length);
      rv_txt = word.substr(rv,word.length);
    }

    var word_after0 = word;

    var array1 = new Array('anza', 'anzas', 'ico', 'ica', 'icos', 'icas',
'ismo', 'ismos', 'able', 'ables', 'ible', 'ibles', 'ista', 'istas', 'oso',
'osa', 'osos', 'osas', 'amiento', 'amientos', 'imiento', 'imientos');
    var array2 = new Array('icadora', 'icador', 'icación', 'icadoras',
'icadores', 'icaciones', 'icante', 'icantes', 'icancia', 'icancias', 'adora',
'ador', 'ación', 'adoras', 'adores', 'aciones', 'ante', 'antes', 'ancia',
'ancias');
    var array3 = new Array('logía', 'logías');
    var array4 = new Array('ución', 'uciones');
    var array5 = new Array('encia', 'encias');
    var array6 = new Array('ativamente', 'ivamente', 'osamente', 'icamente',
'adamente');
    var array7 = new Array('amente');
    var array8 = new Array('antemente', 'ablemente', 'iblemente', 'mente');
    var array9= new Array('abilidad', 'abilidades', 'icidad', 'icidades',
'ividad', 'ividades', 'idad', 'idades');
    var array10 = new Array('ativa', 'ativo', 'ativas', 'ativos', 'iva',
'ivo', 'ivas', 'ivos');

    if ((suf = endsinArr(r2_txt, array1)) != '') {
```

```
      word = word.substr(0,(word.length-suf.length));// substr($word,0, -
strlen($suf));
    } else if ((suf = endsinArr(r2_txt, array2)) != '') {
      word = word.substr(0,(word.length-suf.length));// substr($word,0, -
strlen($suf));
    } else if ((suf = endsinArr(r2_txt, array3)) != '') {
      //word = word.concat(word.substr(0,(word.length-suf.length)),'log');
//$word = substr($word,0, -strlen($suf)) . 'log';
      word = word.substr(0,(word.length-suf.length)) + 'log';
    } else if ((suf = endsinArr(r2_txt, array4)) != '') {
      //word = word.concat(word.substr(0,(word.length-suf.length)),'u');
//$word = substr($word,0, -strlen($suf)) . 'u';
      word = word.substr(0,(word.length-suf.length)) + 'u';
    } else if ((suf = endsinArr(r2_txt, array5)) != '') {
      //word = word.concat(word.substr(0,(word.length-suf.length)),'ente');
//$word = substr($word,0, -strlen($suf)) . 'ente';
      word = word.substr(0,(word.length-suf.length)) + 'ente';
    } else if ((suf = endsinArr(r2_txt, array6)) != '') {
      word = word.substr(0,(word.length-suf.length));
    } else if ((suf = endsinArr(r1_txt, array7)) != '') {
      word = word.substr(0,(word.length-suf.length));
    } else if ((suf = endsinArr(r2_txt, array8)) != '') {
      word = word.substr(0,(word.length-suf.length));
    } else if ((suf = endsinArr(r2_txt, array9)) != '') {
      word = word.substr(0,(word.length-suf.length));
    } else if ((suf = endsinArr(r2_txt, array10)) != '') {
      word = word.substr(0,(word.length-suf.length));
    }

    if (word != word_after0) {
      r1_txt = word.substr(r1,word.length); //$r1_txt = substr($word,$r1);
      r2_txt = word.substr(r2,word.length); //$r2_txt = substr($word,$r2);
      rv_txt = word.substr(rv,word.length); //$rv_txt = substr($word,$rv);
    }
    var word_after1 = word;

    var array11 = new Array('ya', 'ye', 'yan', 'yen', 'yeron', 'yendo', 'yo',
'yó', 'yas', 'yes', 'yais', 'yamos');

    if (word_after0 == word_after1) {
      //Hacemos el paso 2a si no quitamos ninguna terminación en el paso 1.
      if (((suf = endsinArr(rv_txt, array11))!=
'')&&((word.substr(((word.length-suf.length)-1),(((word.length-suf.length)-
1)+1)))==('u'))) {
      //((suf = endsinArr(rv_txt, var array11 = new Array('ya', 'ye', 'yan',
'yen', 'yeron', 'yendo', 'yo', 'yó', 'yas', 'yes', 'yais', 'yamos'))) != ''
&&
      //(word.substr(((word.length-suf.length)-1),(((word.length-suf.length)-
1)+1)) == 'u' ) {//(substr($word,-strlen($suf)-1,1) == 'u')) {
        word = word.substr(0,(word.length-suf.length));//$word =
substr($word,0, -strlen($suf));
      }

      if (word != word_after1) {
        r1_txt = word.substr(r1,word.length); //$r1_txt = substr($word,$r1);
        r2_txt = word.substr(r2,word.length); //$r2_txt = substr($word,$r2);
        rv_txt = word.substr(rv,word.length); //$rv_txt = substr($word,$rv);
```

```
      }
      var word_after2a = word;

      // Hacemos el paso 2b si hicimos el paso 2a, pero fallo al quitar el
sufijo.
      var array12 = new Array('en', 'es', 'éis', 'emos');
      var array13 = new Array('arían', 'arías', 'arán', 'arás', 'aríais',
'aría', 'aréis', 'aríamos', 'aremos', 'ará', 'aré', 'erían', 'erías', 'erán',
'erás', 'eríais', 'ería', 'eréis', 'eríamos', 'eremos', 'erá', 'eré',
'irían', 'irías', 'irán', 'irás', 'iríais', 'iría', 'iréis', 'iríamos',
'iremos', 'irá', 'iré', 'aba', 'ada', 'ida', 'ía', 'ara', 'iera', 'ad', 'ed',
'id', 'ase', 'iese', 'aste', 'iste', 'an', 'aban', 'ían', 'aran', 'ieran',
'asen', 'iesen', 'aron', 'ieron', 'ado', 'ido', 'ando', 'iendo', 'ió', 'ar',
'er', 'ir', 'as', 'abas', 'adas', 'idas', 'ías', 'aras', 'ieras', 'ases',
'ieses', 'ís', 'áis', 'abais', 'íais', 'arais', 'ierais', '  aseis',
'ieseis', 'asteis', 'isteis', 'ados', 'idos', 'amos', 'ábamos', 'íamos',
'imos', 'áramos', 'iéramos', 'iésemos', 'ásemos');
      if (word_after2a == word_after1) {
        if ((suf = endsinArr(rv_txt, array12)) != '') {
          word = word.substr(0,(word.length-suf.length)); //$word =
substr($word,0, -strlen($suf));
          if (endsin(word, 'gu')) {
            word = word.substr(0,word.length-1);  //$word = substr($word,0,-
1);
          }
        } else if ((suf = endsinArr(rv_txt, array13)) != '') {
          word = word.substr(0,word.length-suf.length);  //$word =
substr($word,0, -strlen($suf));
        }
      }
    }

    // El paso 3 lo hacemos siempre.
    var array14 = new Array('os', 'a', 'o', 'á', 'í', 'ó');
    var array15 = new Array('e','é');
    r1_txt = word.substr(r1,word.length); //$r1_txt = substr($word,$r1);
    r2_txt = word.substr(r2,word.length); //$r2_txt = substr($word,$r2);
    rv_txt = word.substr(rv,word.length); //$rv_txt = substr($word,$rv);

    if ((suf = endsinArr(rv_txt, array14)) != '') {
      word = word.substr(0,word.length-suf.length);//$word = substr($word,0,
-strlen($suf));
    } else if ((suf = endsinArr(rv_txt , array15)) != '') {
      word = word.substr(0,word.length-1); //$word = substr($word,0,-1);
      rv_txt = word.substr(rv,word.length);//$rv_txt = substr($word,$rv);
      if (endsin(rv_txt,'u') && endsin(word,'gu')) {
        word = word.substr(0,word.length-1);//$word = substr($word,0,-1);
      }
    }
    word = word.toUpperCase();
    return removeAccent(word);
  }

function stemmer(frase){
  var nuevafrase = "";
  var jfrase =  new Object();
  jfrase = new Array();
```

```
  //frase = document.getElementById('share_description').value;
  frase_aux = frase.split("");
  for (var i = 0; i < frase_aux.length; i++){
    if ((frase_aux[i]==",")||(frase_aux[i]==".")){frase_aux[i]="";}//Pasar a
mayusculas(afecta mas a la funcion stemm).
  }
    frase = frase_aux.join("");
    frase = frase.split(" ");
    //frase = frase.toUpperCase();
    for (var i = 0; i < frase.length; i++){
      //nuevafrase += stemm(frase[i]);
      jfrase[i] = stemm(frase[i]);
      //document.getElementById('result').innerHTML = nuevafrase;//Guardar
las nuevas KEYWORDS en JSON.

    }
    //jfrase = JSON.stringify(jfrase);
    //document.getElementById('result').innerHTML =jfrase;
    //quitar espacios blancos

    //quitar stop words. Hacer pruebas luego con las que tienen acentos
    var stopwords = ['a',
                     'un',
                     'una',
                     'unas',
                     'unos',
                     'uno',
                     'sobre',
                     'de',
                     'todo',
                     'también',
                     'tras',
                     'otro',
                     'algún',
                     'alguno',
                     'alguna',
                     'algunos',
                     'algunas',
                     'ser',
                     'es',
                     'soy',
                     'eres',
                     'somos',
                     'sois',
                     'esto',
                     'estoy',
                     'esta',
                     'estamos',
                     'estais',
                     'estan',
                     'como',
                     'en',
                     'para',
                     'atras',
                     'porque',
                     'por qué',
                     'estado',
```

```
'estaba',
'ante',
'antes',
'siendo',
'ambos',
'pero',
'por',
'no',
'poder',
'sal',
'al',
'puede',
'puedo',
'más',
'ya',
'le',
'o',
'me',
'hasta',
'durante',
'ni',
'ese',
'contra',
'eso',
'mí',
'mi',
'el',
'él',
'podemos',
'podeis',
'pueden',
'fui',
'fue',
'fuimos',
'fueron',
'hacer',
'hago',
'hace',
'hacemos',
'haceis',
'hacen',
'cada',
'fin',
'incluso',
'primero',
'desde',
'conseguir',
'consigo',
'consigue',
'consigues',
'conseguimos',
'consiguen',
'ir',
'voy',
'va',
'vamos',
'vais',
```

```
'van',
'vaya',
'gueno',
'ha',
'tener',
'tengo',
'tiene',
'tenemos',
'teneis',
'tienen',
'la',
'lo',
'las',
'los',
'su',
'aqui',
'mio',
'poco',
'tu',
'tú',
'te',
'si',
'sí',
'tuyo',
'ellos',
'ella',
'y',
'del',
'se',
'ellas',
'nos',
'nosotros',
'vosotros',
'vosotras',
'si',
'dentro',
'solo',
'solamente',
'saber',
'sabes',
'sabe',
'sabemos',
'sabeis',
'saben',
'ultimo',
'largo',
'bastante',
'haces',
'muchos',
'aquellos',
'aquellas',
'sus',
'entonces',
'tiempo',
'verdad',
'verdadero',
'verdadera',
```

```
'cierto',
'ciertos',
'cierta',
'ciertas',
'intentar',
'intento',
'intenta',
'intentas',
'intentamos',
'intentais',
'intentan',
'dos',
'bajo',
'arriba',
'encima',
'usar',
'uso',
'usas',
'usa',
'usamos',
'usais',
'usan',
'emplear',
'empleo',
'empleas',
'emplean',
'ampleamos',
'empleais',
'valor',
'muy',
'era',
'eras',
'eramos',
'eran',
'modo',
'bien',
'cual',
'cuando',
'donde',
'mientras',
'quien',
'con',
'entre',
'sin',
'trabajo',
'trabajar',
'trabajas',
'trabaja',
'trabajamos',
'trabajais',
'trabajan',
'podria',
'podrias',
'podriamos',
'podrian',
'podriais',
'yo',
```

```
                'aquel',
                'que',
                '1','2','3','4','5','6','7','8','9','0']
for(i=0;i<jfrase.length;i++){
  for(j=0;j<stopwords.length;j++){
    if(jfrase[i]==stopwords[j]){
      jfrase.splice(i,1);
    }
  }
}
for(i=0;i<jfrase.length;i++){
  jfrase[i]='"'+jfrase[i]+'"';
}
//jfrase = JSON.stringify(jfrase);
//jfrase=jfrase.replace(/{/g,'');
//jfrase=jfrase.replace(/}/g,'');
jfrase="["+jfrase+"]";
//alert(jfrase);
return jfrase;
```

Thumbnails

El cálculo de los thubnails se lleva a cabo en las siguientes funciones, detecta que se va a subit una imagen, la lee y gracias al manejador de eventos de JavaScript genera el canvas sobre el que pintar el thumbnail de la imagen y subirlo a S3:

```
function uploadObjetob(){
var fileChooser = document.getElementById('filech');
var button = document.getElementById('uploadbtn');

AWS.config.region = 'eu-west-1';
AWS.config.credentials = ({accessKeyId:'',secretAccessKey:''})

 var file = fileChooser.files[0];//Filechooser es un input de tipo file.

    if (file) {
      nuevakey=diractual+file.name;
      var bucket = new AWS.S3({params: {Bucket: 'multimedia-sharing', Key:
nuevakey, ContentType: file.type, Body: file, ACL: 'public-read'}});
      bucket.putObject(function (err, data) {
        if (err){}
        else {
            createPanel(diractual);
            var imageType =  /image.*/;
            if (file.type.match(imageType)) {//Comprueba que sea una imagen
              nombreImg=nuevakey;
              //alert("Procedo a crear el thumbnail")
              var reader = new FileReader();//FileReader que controla la
carga de la imagen y trabaja con ella.

              if (reader != null) {

                reader.onload = GetThumbnail;
                reader.readAsDataURL(file);
              }
            }
          }
       // }
      });
    }
    else {}
//continua en la funcion getthumbnail.

}//end uploadObjeto

function GetThumbnail(e) {//e ese el file, el objeto que hemos cargado y
vamos a trabajar con el
  var myCan = document.createElement('canvas');
  var img = new Image();
  img.src = e.target.result;
  img.onload = function () {

    myCan.id = "myTempCanvas";
    relation=img.width/img.height
```

```
    myCan.width = Math.max(128,img.width/4);
    myCan.height = myCan.width/relation;
    if (myCan.getContext) {
      var cntxt = myCan.getContext("2d");
      cntxt.drawImage(img, 0, 0, myCan.width, myCan.height);
      var dataURL = myCan.toDataURL();

      if (dataURL != null && dataURL != undefined) {
        var nImg = document.createElement('img');
        nImg.src = dataURL;
        uploadThumbnail(nImg);
        }
      else
      alert('unable to get context');
      }
  }
}
function dataURItoBlob(dataURI) {
    // convert base64/URLEncoded data component to raw binary data held in a
string
    var byteString;
    if (dataURI.split(',')[0].indexOf('base64') >= 0)
        byteString = atob(dataURI.split(',')[1]);
    else
        byteString = unescape(dataURI.split(',')[1]);

    // separate out the mime component
    var mimeString = dataURI.split(',')[0].split(':')[1].split(';')[0];

    // write the bytes of the string to a typed array
    var ia = new Uint8Array(byteString.length);
    for (var i = 0; i < byteString.length; i++) {
        ia[i] = byteString.charCodeAt(i);
    }

    return new Blob([ia], {type:mimeString});
}

function uploadThumbnail(img){
AWS.config.region = 'eu-west-1';
AWS.config.credentials = ({accessKeyId:'',secretAccessKey:''})

      nuevakey=nombreImg;
      punto = nuevakey.indexOf(".");
      nuevakey= nuevakey.replace(".","thb");

      var bucket = new AWS.S3({params: {Bucket: 'multimedia-sharing', Key:
nuevakey, Body: dataURItoBlob(img.src), ACL: 'public-read'}});
      bucket.putObject(function (err, data) {
        if (err){"Fallo al subir el thumbnail"}
        else {
          //alert("thumbnail creado")
            createPanel(diractual);
          }
      });
```

Construcción del índice inverso

El cáclculo de la tabla con el índice inverso se realiza en este script:

```
# -*- coding: cp1252 -*-
import boto
from boto.sts import STSConnection
import boto.sdb
import json


####################GET PROFILES#########################
def buildInvIndex():
    #Conexion a simpledb y query
    sdb=boto.sdb.connect_to_region('eu-west-1', aws_access_key_id='',
aws_secret_access_key='')

    #borramos el dominio que vamos a recrear. SimpleDB no tiene una funcion
"truncate_table" de modo que lo hacemos asi
    #-------------------------------------------------------------------------
----------------------------------------
    try:
        sdb.delete_domain('MMS_RESOURCE_INVERSE')
        print "el dominio de tabla inversa ha sido borrado"
    except:
        print "el dominio de tabla inversa no existe. será creado"
    sdb.create_domain('MMS_RESOURCE_INVERSE')
    print "el dominio de tabla inversa ha sido creado"

    #ahora conseguimos un objeto de tipo dominio para poder lanzar queries
contra nuestra tabla original. nos viene bien
    #usar el nuevo dominio para tambien poder lanzar inserciones en el nuevo
    dom=sdb.get_domain('MMS_RESOURCE_INVERSE')


    #exploramos la tabla MMS_RESOURCE_TABLE y construimos varias tuplas en
MMS_RESOURCE_INVERSE, una por cada keyword
    #-------------------------------------------------------------------------
----------------------------------------
    last_item_name="0"
    fin=False
    num_vuelta=0

    tuplas_insertadas=0;#nos va a servir de item_name para la nueva tabla
    operaciones_insercion=0 # para ver cuantas veces invocamos a
batch_put_attributes. será igual al numero de tuplas de MMS_RESOURCE_TABLE
    queries_realizadas=0# queries hechas contra MMS_RESOURCE_TABLE

    # query_size=tuplas de MMS_RESOURCE_TABLE leidas en cada iteracion. Este
numero es truncado por simpledb segun los datos recogidos
    # por ello a priori no podemos saber cuantos va a leer, pero 100 siempre
está por encima, de modo que leemos lo máximo
    query_size=100

    while fin==False:
```

```
        if num_vuelta==0:
            query='select KEYWORDS,MMS_RANK from MMS_RESOURCE_TABLE limit
'+str(query_size)
        else: query='select KEYWORDS,MMS_RANK from MMS_RESOURCE_TABLE where
itemName() > "'+ last_item_name + '"' + 'limit '+str(query_size)

        queries_realizadas+=1;

        rs=dom.select(query, max_items=str(query_size)) # rs es "result set"
        '''#el max items es para que no ignore el limit'''
        '''#Tengo que anidar la query y cambiar la logica de este programa'''
        elems_rs=0#numero de elementos en rs

        #items={} # creamos un diccionario
        for j in rs:
            elems_rs+=1
            #print elems_rs
            print j
            last_item_name=j.name#PRUEBA
            #print last_item_name
            # ahora debo recorrer j.KEYWORDS y meter una tupla por cada una
en MMS_RESOURCE_INVERSE
            #-------------------------------------------------------------------
----------------------
            lista=json.loads(j['KEYWORDS'])
            print "lista:=",lista
            #items = {'item1':{'attr1':'val1'},'item2':{'attr2':'val2'}}
            items={} # creamos un diccionario

            for i in lista:

                #print "tupla a insertar :"
                print "           keyword :", i
                print "           mms_item :", last_item_name
                print "           mms_rank :", j['MMS_RANK']
                #creamos un diccionario
                item={}
                item['KEYWORD']=i
                item['MMS_ITEM']=last_item_name
                item['MMS_RANK']= j['MMS_RANK']
                #inserto el diccionario en la lista de diccionarios
                items[tuplas_insertadas]=item
                tuplas_insertadas+=1

            #ahora me queda insertar las tuplas en MMS_RESOURCE_INVERSE
            #-----------------------------------------------------------
            #print items
            # estas son todas las tuplas que genera una row de
MMS_RESOURCE_TABLE
            #esta funcion como mucho permite 25 tuplas, de modo que un
recurso no deberia tener mas de 25 keywords o esto dará error
            dom.batch_put_attributes(items)
            operaciones_insercion+=1

        if elems_rs==0:
            fin=True
```

```
        num_vuelta+=1
        print "-----------------------------------------------------------------
-----"
        print "elementos leidos en esta iteracion"    , elems_rs

    print "tuplas insertadas:", tuplas_insertadas
    print "operaciones de insercion (batch_put_attributes):",
operaciones_insercion
    print "iteraciones (queries contra MMS_RESOURCE_TABLE):",
queries_realizadas

    return "hola"


#################### MAIN ########################

print "hello world"

buildInvIndex();
```